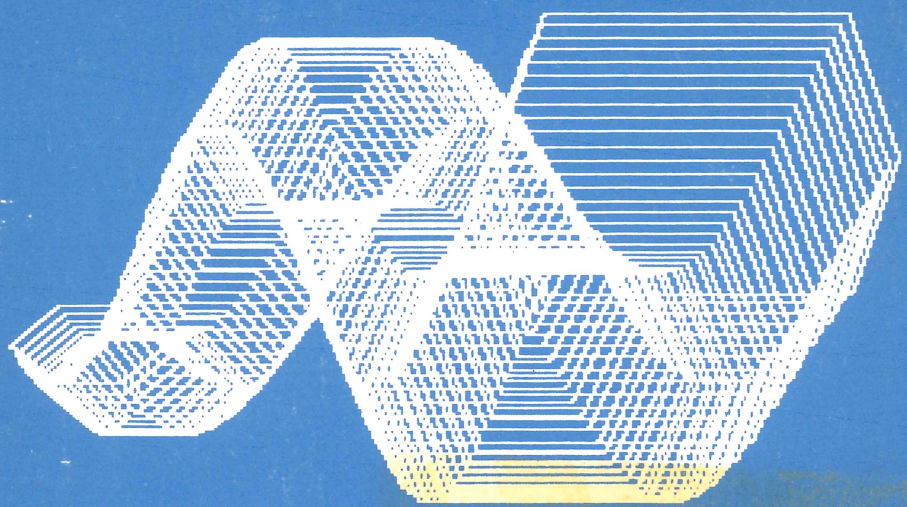
THE FIRST FEW BYTES

Richard F. Devon



THE FIRST FEW BYTES

RICHARD F. DEVON

DEPARTMENT OF ENGINEERING GRAPHICS

THE PENNSYLVANIA STATE UNIVERSITY

THIS MANUAL IS INTENDED TO PROVIDE PROGRAMMING LITERACY IN APPLESOFT BASIC AND FILE MANAGEMENT LITERACY (DOS) FOR THE APPLE II+ OR IIE. IT IS INTENDED FOR STUDENTS IN COLLEGE OR COLLEGE TRACK IN HIGH SCHOOL WHO HAVE A MATH OR SCIENCE ORIENTATION AND WHO LIKE TO LEARN QUICKLY.

Apple II and Applesoft are registered trademarks of Apple Computer Inc., 10260 Bandley Drive, Cupertino, California 95014.

KENDALL/HUNT
PUBLISHING COMPANY
Dubuque, Iowa

This edition has been printed directly from the author's manuscript copy.

Copyright © 1984 by Kendall/Hunt Publishing Company

ISBN 0–8403–3250–5

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the copyright owner.

Printed in the United States of America

B 403250 01

CONTENTS

Preface.....1

PART I
THE LITERACY SEQUENCE

Introduction to Microcomputers. 4

1. Immediate Execution.....12

2. Deferred Execution.....15

3. The FOR / NEXT Loop. 20

4. Numbers, Variables and Functions. 24

5. Give Me Your Numbers: Data And Other Inputs. 30

6. The Elements of Style: Editing and Formating. 34

7. Low Resolution Graphics. 40

8. High Resolution Graphics. 45

9. Introduction to Intermediate Programming. 52

10. An Introduction to Computer Graphics.....58

PART II
REFERENCES

11. DOS: The File Management System.....68

12. Bibliography.....74

13. Periodicals.....75

14. Glossary.....78

PREFACE

This guide was written as I learnt the information it now contains. As I have progressed I have tried to keep the manual dedicated to answering the original questions on the assumption that others will approach the Apple II similarly. This assumption seems reasonable since the manual is not written for everybody, but for a dedicated population, viz., people with a bent towards math and science who want to learn quickly.

In addition to writing the manual from the learners point of view with respect to the questions asked, I also had the learner in mind in another way. This manual is not comprehensive, encyclopedic, theoretical or technical. Nor is it a compendium of program listings. The approach is practical, and what you don't need to know in order to act is not included. The result is that knowledge and action are closely related. Therefore, the rewards of accomplishment are frequent and should be a continuous source of motivation as the learner proceeds through the manual.

There is a great deal more to know than is covered, or even suggested, in this manual. The manual is only intended to facilitate the acquisition of literacy in Applesoft programming and in file management (DOS). A short bibliography and a list of periodicals are provided as opportunities for further learning.

The manual was written, then, with the following purposes in mind:-

1. To produce a curriculum appropriate for the hardware system, viz., Apple II+s, or IIes.

2. To set an instructional pace appropriate for first year engineering or science students, or college track math and science high school students.

3. To use an instructional mode aimed at maintaining a high level of motivation in the students by teaching them how to program immediately, thereby inducing feelings of power, pleasure and satisfaction. The myriad subtleties of programming are left for later acquisition.

4. To use an instructional mode which stresses learning in context and through doing, rather than by definition and routinized information processing. For example, the TRACE command is introduced to trace an error in a program, and that is when, and how, it is learned. A corollary of this is that the manual keeps the student on the keyboard, and little, if any, lecturing or other reading is required for the duration of the literacy acquisition process.

SCHEDULING

A recommended schedule for using the manual is as follows:-

ACTIVITY	FORMAL CLASS HOURS	EXTRA HOURS
Sessions 1-9	12	3
Terminology Quiz	1/3	
Session 10	4	1
Programming Quiz	2/3	
Project	4	10

A total of about 35 hours should be adequate for the completion of the manual including the project. Approximately two-thirds of this time is best spent in a formal class situation, but the manual can easily be used alone without the assistance of an instructor. Instructors need to be warned that an evaluation system is introduced at the end of Session One.

Projects are not hard to think of. An example which led to the cover design is:-

PROBLEM. Construct a hexagon that may be translated anywhere on the screen, and which may be rotated about any point. Include a demonstration that moves the shape, with or without erasing, along a curvilinear path.

THE APPLE II+ AND THE APPLE IIe

This manual was written on an Apple II+. It is quite appropriate for use with the IIe also. The relevant differences are few. The most important differences affect the use of the Apple as a word processor. This is particularly true if the IIe has an 80-column card. Word processing is a very useful function of the Apple, but it is not covered in this manual.

The IIe has upper and lower case letters, but the lower case letters cannot be used in programming. The IIe has a different keyboard with several extra keys, none of which affect programming at the level of this manual. The IIe has 64K memory, but that does not affect the relatively small memory demands of the programs in, and implied by, this manual. Further, the II+ can be boosted to 64K for well under $100. Also, the II+ uses small standard chips and is probably cheaper to maintain, which is of interest to some people.

The only difference which IIe users will need to make an adjustment for in this manual occurs if they have an 80 column card. In this case the format commands such as TAB and HTAB will change to accomodate 80 columns rather than the 40 columns of the II+ and the IIe without the 80 column card. Where the 80 column card is present, the user still has option to use the 40 column mode if they prefer.

INTRODUCTION

THE LEARNING SYSTEM

This manual is written for use with an Appple II+ or IIe. The language used is the Applesoft version of BASIC.

LEARNING SOURCES

To facilitate the speedy acquisition of microcomputer literacy, you should have many sources of information readily available. Some of the more accessible of which are listed below.

1. This manual.
2. Yourself.
3. Other students.
4. Your instructor.
5. Apple manuals. There are many of these. The Apple II+ was,and still is sometimes, sold with the Applesoft Tutorial, the Basic Programming Reference Manual and the Apple II Reference Manual. Versions of these are available for the Apple IIe, but usually only the IIe Reference Manual comes with the purchase. The Tutorial is of little use to people using this manual. The reference manual is the source for technical data. The programming manual is the source for Applesoft. The DOS Manual (which is very good) is provided with Apple II disk drives and it is the source for the file management system. Ask your dealer about other Apple manuals available.

MICROCOMPUTERS

Computers are important because they can do calculations in extremely small periods of time, and because they can store massive amounts of information in very small spaces. Computers come in three basic sizes: main frame, or macrocomputer, mid-size or minicomputer and personal, or microcomputer. Although large scale programming and data processing,and sophisticated graphics, can only be done on the larger computers, the rapidly improving microcomputers have also become extremely popular in industry, business, research, education and the home. They are used for analysis, accounting, word-processing, file-keeping, communication and graphics. Some people even play games on them. Microcomputers have become very important recently for several reasons: their costs have fallen to the mass market range, their capacities have increased tremendously and they are now powerful tools, and because they have a stand-alone character. This last point is important because it means the user is not dependent on a large system which may be vulnerable to breakdowns, shutdowns

for holidays, user saturation delays and high costs. Further, micros are more user friendly than macros, and some people prefer the security and privacy of having their own system.

The field of computers is characterised by very rapidly changing technology. Until 1968, the largest computer at Penn State was an IBM 7074. The Apple II+, while much smaller physically, is faster and has a larger memory than the 7074. The main frame of 15-20 years ago may not be worth a $100 on todays microcomputer market.

SERVANT TROUBLE

The computer is a wonderful worker: incredibly fast and almost as smart as you are. However, in order to get the computer to work for you, you must learn to speak to it in precisely the terms it desires. One word out of place, one comma out of place, and you have a strike on your hands. Worse still, it will make a rude noise and accuse you of making a mistake - ?SYNTAX ERROR is the usual charge. Don't waste time arguing with it - you can't win. A thousand mistakes lie between you and microcomputer competency: the sooner you make them, the sooner you will get there. In the early exercises, after getting ?SYNTAX ERROR, just start again. The Apple bears no grudges (usually) and will be waiting for a fresh attempt.

THE KEYBOARD

The keyboard of the Apple is very similar to that of a typewriter. Study it and note the differences. Some of the more important differences are indicated here.

ESC The escape key which allows you to escape the writing mode and edit the program you are in.

CTRL The control key when held down and followed by C will halt the execution of a program except when an input command is operating. If the control key is held down during a program listing or print output the letter S may be used as a toggle (stop/start) switch.

RESET The reset key is very powerful. It will stop program execution in almost all cases. The program will be left intact ready to be run again. The option that requires the CONTROL button to be simultaneously held down when using RESET may be activated by a switch inside the computer (under the left end of the keyboard).

RETURN When you type a message into the computer, it will appear on the screen as you do. It will not, however, be sent to the computer until you press RETURN. The message may be several screen lines long, but the only symbols sent to the computer will be those to the left of the flashing cursor at the moment when you press RETURN.

Arrows. Backspace and forwardspace arrows are on the right of the keyboard. If you spot an error before you hit RETURN, you may backspace over your line, correct the error and forwardspace again without erasure. In fact, the backspace key does erase, but the forward space key retypes again and is called the retype key. The spacebar works for erasure. Make sure that the cursor has been moved back to the right end of the line before you hit RETURN.

REPT Hold down a letter, say, R. Now hold down the REPT key, also. The letter R is now being rapidly repeated. To stop, raise either, or both, keys.

Note that all the keys may be programmed for specific functions in a particular program.

TURNING ON

The Apple is turned on by a toggle switch at the back (left side) of the computer. The monitor switch is in the front. If a disk drive is connected, it will turn on automatically when the computer does and if their is no disk in the drive you must press RESET to stop the drive's abortive attempt to read a disk. Some security systems consist of a frame that is bolted to the table to prevent the Apple from being easily removed. In this case one switch may turn on computer, monitor, drive and a cooling fan.

THE SYSTEM

A computer is always part of a system. A system typically includes means of accepting input in the form of messages, data and programs, of producing output in the form of messages, data or pictures, and of storing programs and data. The keyboard is a means of input, the screen is an output mode and floppy disks driven by disk drives are a way of storing programs and are used for both input and output.

A stand alone system refers to one that consists of, at least, a microcomputer with a keyboard input and one form of output (usually screen or paper). Typically, a system will consist of three forms of output, a monitor, a disk drive and a printer, and two forms of input, a keyboard and a disk drive. Other forms of input are graphics tablets and light pens. The disks represent permanent memory. Printed or plotted output on paper (hard copy) represents permanent output. An increasingly popular system is a local area network (LAN) where microcomputers are networked together and share peripherals such as printers or a hard disk. (A hard disk is much more expensive than a floppy disk, although it comes with its own drive. It is usually fixed in place, but it is possible to get removable hard disks. The memory capacity is very much larger than a floppy disk and the execution speed is much faster.)

There are many variations possible with stand alone systems. Multiple disk drives may be used. A plotter which is designed for graphics as opposed to printers, which accomodate graphics with varying degrees of success, can be used to complement or replace a printer. Even the independence of the system can be made a variable by using a modem to connect the computer to the telephone system. In this way computers can be linked together and programs, data and messages transferred. Alternatively, the microcomputer may be hooked up to a main frame computer via the modem and used as a terminal

The above only describes the hardware of the system, that is, its physical parts. The system also requires a means of communication between these parts and a means of handling information within the computer itself, that is, software. The most important of the former is the disk operating system (DOS). The latter involves a fairly complex set of operating programs and codes which are structured so that the average user only need learn Applesoft to operate the computer.

The operating programs and Applesoft are stored in the permanent memory of the Apple known as ROM. This is the read only memory because although you can read (take) information from it you cannot write on (change) it. The programming that you will do will take place in the RAM - random access memory. The RAM is dynamic and the contents are lost when the computer is turned off, which is why disks are used to save the programs on.

This manual is intended to bring the reader to programming literacy in Applesoft and to user literacy in DOS. While the competency attained will not be expert, it should be of great utility and adequate to facilitate continued learning.

The most used commands of the disk operating system (DOS) are those which control the saving, running, cataloging and deleting of programs. Even without knowing how to write a program you can do a great deal using only the DOS commands, because these allow you to run any program written for the Apple II of which you have a copy.

THE DISK

At any given time there can only be one program in the RAM of the Apple computer. Disks, hard or floppy, are the most common place for storing all the other programs. The disk drive will read information from, and write information on, the magnetic surface of the disk (so keep your disks away from from magnetic fields, e.g., TV sets).

There are many forms of disks. The Apple II+ and IIe typically use a 5-1/4 inch floppy disk. This disk consists of 35 concentric tracks each of which has 16 sectors. Therefore there are 560 sectors on a disk, and, since each sector can contain 256

characters, the whole disk can store 143,360 characters which is equivalent to about 80 double spaced typewritten pages.

The disk must loaded into the disk drive. There are two ways to do this.

A. 1. Turn the Apple off.
   2. Gently load the disk in the disk drive (label upwards and outwards).
   3. Close the disk drive.
   4. Turn the Apple on.

B. Assuming the Apple is already on:-

   1. Load the disk in the disk drive.
   2. Close the disk drive.
   3. Type PR#6 and press RETURN.

Both these methods effectively boot (connect) the disk to the computer.

Note that RETURN must be pressed following any command.

Some disks are programmed to come up running with instructions and information for the user. Generally, if no catalog of contents appears, you should type CATALOG to find out what programs are on the disk. To run a program listed on a disk, type RUN followed by the program name.

Remember, booting a disk will erase current memory. If a disk is already booted, you may save the program in current memory with the SAVE command followed by the name you wish to give the program. If it is already on disk there is no point in saving it unless you have modified it.

DOS COMMANDS.

LOCATORS. The Apple has eight (0-7) slots for interfacing with other equipment such as disk-drives or printers. When you wish to interface through a particular slot you may have to append the slot number, e.g., ,S6. It is only necessary to do this when you change slots. It is not necessary to specify the slot (or drive) that you are already in. Two disk-drives can be connected to one slot. This, too, must be appended, e.g., ,S6,D1 or ,S6,D2 or simply ,D1 or ,D2 if you are not changing slots. Slot #0 is the keyboard and it is called when the user wishes to return to the keyboard from, say, a printer. It is also where a 16K language card for enhancing the memory capacity is installed. All the other slots may be assigned to peripheral devices according to the wishes of the user. However, it is now standard practice to put the disk drive in slot #6 and the printer in slot #1.

In general, all of the following commands may have ,Sn,Dm appended to them as location finders, where n and m are the slot

and drive numbers, respectively. The order of S and D is not important. For a typical stand alone system with two disk drives only the D will change, and if you only have one disk drive you will not need any appended locators.

To boot, or connect to, a peripheral device, type PR#n, where n is the slot # the device is connected to. If the slot has two disk drives, drive #1 will boot.

SAVE. When you have written a program, you may save it by giving it a name, such as VECTOR, and typing :-

```
SAVE VECTOR,S6,D1
```

The actual slot and drive numbers you will use, if any, may be different to this. Don't forget the comma after the program name or S6 will become part of the name saved in D1 of the slot you are in at the time. The name may not be more than 30 characters in length, and the first two characters must be letters.

Note. The program that you save will replace any program of exactly the same name that is already on a disk in that slot and drive location. A program that is run or loaded from a disk under one name may be given a new name when saving. There will then be two copies each with a different name.

CATALOG. This will give you a directory of any programs that are on the disk that you are currently accessing.

The directory may look something like this:-

```
A 007 HELLO
A 015 VECTOR
B 004 DOSM
I 028 GRAPH
T 018 LAB
```

The A stands for Applesoft and when you save the programs you write they will be listed with an A because they will be in Applesoft. The B stands for binary and it means that the program was written in assembly language. Assembly language is a low level language, meaning that it is structured to fit the architecture (hardware organization) of the computer. Because of this, assembly is hard to write programs in but, when you do, they run much more quickly. The I stands for integer basic and the T stands for text (a text or data) file. To work with B, I or T files you should consult the DOS manual.

The numbers in the directory listing refer to the number of sectors used by the program.

RUN. This will run any program that is in current memory. It can also be used to run the programs on a disk by appending the name of the program you want to run, e.g., RUN VECTOR. To run a binary file (program) you must use BRUN rather than RUN.

LOAD. This command loads a program from the disk to the current memory of the computer without running it. Example:-

```
LOAD ACORN,S6,D1
```

DELETE. Deletes a program from a disk. Example:-

```
DELETE ACORN,S6,D1
```

RENAME. You may rename a file (program) by typing RENAME old name, new name. Example:-

```
RENAME ACORN, TRIG
```

If you cataloged after doing this you would find that the program ACORN was no longer listed, whereas TRIG would be, although the program would still the same.

BOOTING A DISK

When a disk boots, a copy of the DOS that is written on the disk is moved into the computers memory. This allows the computer (user) to communicate with the contents of the disk. Two other things happen also. The RAM memory of the computer is otherwise cleared, so you lose any programs that you had in the computer at the time. Secondly, a turnkey program, usually named HELLO, is loaded into the computer and run. This is dicussed below.

INITIALIZING A DISK

A brand new blank floppy disk must be initialized before it can be used. Initializing does two things. It writes the DOS on the first three tracks of the disk with the INIT command. It also provides a turnkey program, that is a program that is always run after DOS is booted. this turnkey program is the one that the user is in after the disk has booted. It may have other names, but HELLO has become the standard. The HELLO program may do nothing more than put a message on the screen. More usefully, it may automatically display the directory.

To initialize a new disk, the easiest way is to load an HELLO program (a version you fancy) from another disk. Then remove the old disk and put in your new disk and type INIT HELLO -and press RETURN. To summarize:-

Put in old disk, type LOAD HELLO
Put in new disk, type INIT HELLO

Your disk is now initialized. If you did to an old disk what you have just done to a new disk, you would have cleared all the existing programs off that disk and left only the new HELLO program. This is useful when you want to recycle a disk. It would

be unfortunate if you did not. As a precaution you should always have at least one backup copy of your programs on another disk. Also, label your disk - and do it with a felt tip pen to avoid damage from pressure or graphite dust.

The other way to initialize a disk is to write your own HELLO program. Example:-

Load DOS into the computer by booting a disk that is already initialized.

Put in the new disk, type

```
5  HOME
10 VTAB 1: HTAB 15
20 PRINT "TRON'S DISK"
30 END
INIT HELLO
```

When this disk boots, the message "TRON'S DISK" will appear at the top of the screen. This is a very simple version of an HELLO program. You will come across many different versions, some of which are quite sophisticated.

After the disk has booted the HELLO program is still in memory and you will need to type NEW before you start writing programs.

For further information about DOS refer to Chapter 11 or the DOS manual.

Before beginning to study programming, we will examine the immediate execution mode of the computer. This is a useful mode both for quick calculations and for examining the operations of a program (see the section at the end of Session Seven). It also serves to underscore what the deferred execution quality of programming means.

SESSION ONE
THE IMMEDIATE EXECUTION MODE

FINDING THE RIGHT WORDS

Type I LOVE YOU and press RETURN. Rude isn't it?

Now, type PRINT" I LOVE YOU " and press RETURN.

If the Apple was rude again, you were not being careful. When you do it correctly you will realize that you have successfully communicated with the Apple.

TYPE	PRESS	APPLE RESPONSE
2*3=	RETURN	
PRINT 2*3=	RETURN	?SYNTAX ERROR
PRINT 2*3	RETURN	6
PRINT "2*3"	RETURN	2*3

PRINT is clearly a general command which will complete a calculation (without the =) unless it is in quotes, in which case it will simply print it on the next line. You will see that the asterisk, *, is used for the multiplication sign.

** From now on, pressing RETURN will be up to you to remember.

TYPE	APPLE RESPONSE
A=6	
PRINT A	6
PRINT "A="A	A=6
PRINT "3*A="3*A	3*A=18

This exercise shows that the computer has a memory and it will print the current value for A that is in its memory. It also shows that the PRINT command can operate with variables, such as A, as well as with numbers. Further, we see how the quote function of PRINT can be used to identify the outcomes of PRINT calculations. Next we will look at the way the Apple performs arithmetic. (Note: the ? may be used as shorthand for PRINT.)

ARITHMETIC OPERATORS

The arithmetic operators should be familiar, except for the exponentiation sign, ^, found on the N key (e.g., B squared is typed B^2). Note the order of preference in execution, this is very important.

SYMBOL	FUNCTION	PREFERENCE
+	Addition	4
-	Subtraction	4
*	Multiplication	3
/	Division	3
^	Exponentiation	2
( )	Parentheses	1

Before you type the following commands, enter your prediction and see how well you have learned the order of preference.

TYPE	PREDICTION
PRINT 4+6/3	
PRINT 4*6-3	
PRINT 4*(6-3)	
PRINT 4^2*6	
PRINT 4*6/2^2	
PRINT (4*6/2)^2	

At this point we have already learned to use the Apple as a simple calculator. Next we will learn how to store and sequence commands, that is, how to program.

END SESSION ONE
GOTO PROBLEM SET #1

EVALUATION. There will be no evaluation for Problem Set #1. If you have any difficulty with it, seek help. For subsequent problem sets, save the programs (solutions) and, when the whole set is finished, run them for the instructor. They will be evaluated on a P/F competency basis, and they may be repeated until they are satisfactory. Grades for computer work will be based on projects and quizzes.

PROBLEM SET #1

1. Given that X equals A squared and divided by B, calculate X when A=2.5 and B=9. Print the values of X, A and B with identification, e.g., X= .

2. Next, print the data again using two spaces inside the quotes before each variable, e.g., " X=". This will make the data more readable.

3. In a PRINT statement ; will separate items to be printed side by side. The , will cause the next item to be printed in the next of three columns, viz., col. 1, col.17 or col. 33.

Type
```
A=6
PRINT A,A,A,A,A
PRINT A;TAB(17)A;TAB(33)A;TAB(41)A;TAB(17)A
```

These two formats produce identical results. The TAB method is more versatile, however, since it allows you to start the print out on any of the 40 columns. The , is quick and useful for tabulation. The TAB cannot specify beyond the line limit of 255, which, with 40 columns on the screen, will be almost six and a half lines.

In general, commands like TAB which control monitor output will affect printer output in a different way. To get the desired output will mean consulting the manual for the printer being used.

For the problem, print your name in the middle of the line using the TAB command.

DEFERRED EXECUTION

Devoutly to be wished, and very useful. With a deferred execution system we can sequence and store commands, allowing us to use them whenever we wish and as often as we wish. That is, we can program. To do this we assign numbers to the statements, both as a means of ordering them, and as a means of identifying them so that we can recall them. A program consists of a coherent and ordered group of numbered statements which when run will achieve a desired purpose.

If you make a mistake while typing the following program, you can correct by backspacing if you have not yet hit RETURN. Be sure to forwardspace again, using the right arrow retype key, to the end of the line before you press RETURN. If you have already hit RETURN, then ignore the old line, and, with the same line number, type the correct line statement. The computer will automatically replace the old with the new.

Type
```
NEW
5  REM  TEST PROGRAM
10 A=6
50 PRINT "D="D
30 C=10
40 D=(A^2+B^2+C^2)^.5
20 B=8
```

This program assigns values to A, B and C and then solves D^2=A^2+B^2+C^2. The statements are not listed in a logical sequence, but they are numbered logically. If you type LIST you will see that the computer has put them in the proper numerical order, which, if there are no logical errors, will be the correct execution order. Now type RUN. Fast isn't it?

Single letters have been used as variable names. Longer names may be used, but Applesoft will only discriminate between the first two characters. Avoid Applesoft terms in your choice of variable names.

The REM statement will be ignored by the computer. Its purpose is to identify or explain a program or section of a program for the reader. You should always identify at least the program.

You will notice that the program numbers go up in intervals of 10. This is done to allow opportunities for modifying the program.For example :-

Type
```
45 E=D*100/(A+B+C)
55 PRINT "D="E;"% of(A+B+C)"
RUN
```

SAVE

The program you have just typed into the computer may be run as often as you wish. It is, however, in considerable jeopardy. It is in current memory and if you turn the computer off, or if there is a power failure, the program is lost. If you type NEW, it will be deleted in preparation for a new program. If you load a program from a disk, you will erase current memory. And finally, if you continue writing programs without typing NEW, you will end up with a hybrid that most probably does nothing that you want - if it works at all. To save a program for future use we give it a name and save it. If you are on line with a disk in your disk drive, this is accomplished by typing SAVE PYTHAGORAS (or whatever name you choose). If you are not on line with a disk, that is, if you have not booted DOS, you cannot save your program. You will have to boot, retype and then save.

When you save a program, be careful in choosing a name. If a program with that name is already on the disk, it will be replaced with the new one unless it is specifically protected against this happening (see LOCK in Chapter 11). Applesoft will discriminate between all the characters in program names. Example; BALL1 and BALL2 are both legal and distinct as program names. A name can be up to 30 characters long - a nuisance to type, but legal. The first two characters in a name must be letters. Commas may not be used in names.

LIST

You will use this command very frequently so we will learn it now. If, as you already have, you enter the LIST command, the computer will list all the numbered statements that are in current memory and do so in numerical order. A long listing may be stopped and started again by holding down CONTROL and using the S as a toggle (on/off) switch. A CONTROL C will abort the listing at the moment it is used.

LIST P,Q or LIST P-Q will list only the statements from line number P to line number Q, inclusive. LIST P- will list from, and including, line number P to the end of the program.

FLOW COMMANDS

The rest of this session will focus on some of the commands used to control the flow of the program. Many of these greatly increase the efficiency of execution.

Consider the function Y=X^2 . Suppose we need to write a program

that will calculate the value of Y for X=1 to 20. We could do the following.

```
10 X=1
20 Y=X^2
30 PRINT "Y="Y
40 X=2
And so forth.
```

Or, we could do this. Clear current memory by typing NEW and try it.

Type
```
NEW
10 X=1
20 Y=X^2
30 PRINT"Y="Y
40 X=X+1
50 GOTO 20
```

This illustrates the GOTO command which is used to achieve an execution sequence other than the numerical line order. In our case, we have set up an infinite loop - at least until the numbers generated exceed the Apple's capacity of 1.7*10^38. This is an easy problem to correct, but, before we do, we will use the uncontrolled looping to demonstrate some other commands.

To start the program type RUN
To stop the program press RESET
To restart the program type CONT (or RUN)
To stop hold down CTRL and hit S
To start the program again hit S, C or any other letter
To stop hold down CTRL and hit C
To start the program again type CONT

Repeat these exercises until you are familiar with them. Then stop the program and we will solve the problem permanently.

Type
```
45 IF X>20 GOTO 60
60 END
LIST
RUN
```

You should now have 20 values of Y on the screen from Y=1 to Y=400. A few rounding errors will be apparent. These are produced because the ^ is calculated with a power series. If line 20 read Y=X*X, there would be no rounding errors and the execution would be quicker.

We could have used STOP instead of END. The main difference being that the STOP command stops execution and prints the statement number where it does this (its own).

RELATIONAL OPERATORS

In line 45 we used a relational operator, >, to effect a logical choice. All the usual relational operators are available in Applesoft.

```
          =     EQUALS
          <     LESS THAN
          <=    LESS THAN OR EQUAL TO
          >     GREATER THAN
          >=    GREATER THAN OR EQUAL TO
    >< OR <>    NOT EQUAL TO
```

CONDITIONAL STATEMENTS

Relational operators are used in conditional IF...THEN statements which monitor the program execution. Conditional statements provide the programmer with a great deal of control over program flow. Both AND and OR are permissible in these statements. Here are two further examples:-

```
88 IF A <> B OR C < D THEN X=5
99 IF A > 10 AND A < 50 GOTO 101
```

Note that GOTO does not need to be preceded by THEN.

The WHILE and ELSE conjunctions used in FORTRAN are not permissible in Applesoft.

LOGIC

Using the PRINT statement, Applesoft can be used to test logical propositions. It will print a 1 if the statement is true and a 0 if the statement is false. For example:-

```
10 A=8
20 B=5
30 PRINT A>B
40 PRINT NOT (A>B)
50 PRINT A=5 OR B=5
```

Line 30 is true and a 1 will be printed.
Line 40 is false and a 0 will be printed.
Line 50 is true and a 1 will be printed.

PRECEDENCE

The order of precedence for the arithmetic and logical operators combined is as follows:-

1. ( )
2. NOT

3. ^
4. * /
5. + -
6. > < = >= <= <>
7. AND
8. OR

END SESSION TWO
GOTO PROBLEM SET #2

PROBLEM SET #2

Remember to clear current memory before beginning a new program by typing NEW, and remember to number your statements.

GIVEN. An object at time T=0 is travelling at velocity V1=5.1 ft./sec. The object is subjected to a uniform acceleration of A=8.4 ft/sec^2 for a period of 45.3 seconds.

1. Write a program to find the final velocity V2 where V2=V1+A*T and T=45.3 seconds. Write print statements for V1,V2,A and T.

2. Now, modify your program, using a relational operator and a GOTO command to generate a set of values for V2 corresponding to times of 10,20,30,40 and 50 seconds. Print out values for T and V2 only.

SESSION THREE
THE FOR / NEXT LOOP

The FOR / NEXT loop provides an even better way for programming the Y=X^2 problem than the conditional branching method.

Type
```
NEW
10 FOR X = 1 TO 20
20 Y=X^2
30 PRINT "Y="Y;"  X="X
40 NEXT X
RUN
```

The program establishes a loop beginning at line #10, proceeding through lines #20 and #30, and arriving at line #40. Whereupon the program loops back to line #10 and the sequence is repeated again. On the first loop, the value of X is 1, on the second it is two, and so on, until the last loop when the value of X is 20. The increment to the X value each time is 1 by default because we did not specify the step, but we could have.

Note the spaces left before X= in the PRINT statement. This makes the output more readable.

Type
```
10 FOR X=1 TO 20 STEP 4
```

This will replace your old statement #10. Type LIST if you doubt this.

Now type RUN. You should have generated X values of 1,5, 9,13 and 17. If you want to end exactly on 20 :-

Type
```
10 FOR X=0 TO 20 STEP 4
RUN
```

This should have provided X values of 0,4,8,12,16 and 20.

Now try this.

Type
```
10 FOR X=20 TO 0 STEP -4
RUN
```

Like many things in life, it works both ways.

Although we have used integers, the FOR / NEXT command also accepts fractions. For example:-

Type
```
10 FOR X=.5 TO 2.7 STEP .2
RUN
```

WARNING. If you examine the last output you will notice that the looping ended with X=2.5 instead of 2.7. Due to a rounding problem the Apple does this for some loops when the step is less than .5. One way to handle this is by slightly increasing the upper limit, e.g., use 2.71 instead of 2.7.

Note that we can shorten our program even further by eliminating the variable y.

Type
```
NEW
10 FOR X=1 TO 10 STEP 3
20 PRINT "X="X;"  X^2="X^2
30 NEXT X
```

For one final variation,type:-

```
3 A=0
5 B=20
7 C=4
10 FOR X=A TO B STEP C
LIST
RUN
```

This demonstrates the capacity of the FOR / NEXT command to operate entirely with variables.

The FOR / NEXT loop is one of the most powerful techniques in programming. Obviously, any one program might have more than one such loop scattered throughout the program. Less obviously, these loops may be constructed in parallel in an arrangement known as nesting. The Apple will accept up to 10 levels of nesting, but programs that use more than three or four are rare. We will do an example with only two nested loops.

Type
```
NEW
10 FOR A=1 TO 10 STEP 3
20 FOR X=0 TO 16 STEP 4
30 Y=A*X^2
4o PRINT "Y="Y;"  A="A;"  X="X
50 NEXT X
60 NEXT A
```

Study the flow of the program. For the first value of A, an entire set of X values will be generated. This will happen for each value of A. Since there are 5 values for X and 4 values for A in this program, the program will execute the X loop 20 times. Type RUN. You should get 20 sets of values from Y=0, A=1, X=0 to Y=2560, A=10, and X=16.

The X loop in this program takes place entirely within the A loop. This is why it is called nesting. Loops must never cross. Let's see what happens when they do.

```
Type      50 NEXT A
          60 NEXT X
          LIST
```

Study the program. The computer will run through the A values with X=0 until A reaches its last value of 10. At this point it will arrive at line #60 and be told to get the next X. This it cannot do because the next value of X is at line #20 and line #20 is inside a completed loop and inaccessible. Type RUN and you will get the output values for X=0 only, followed by the error message:-

```
?NEXT WITHOUT FOR ERROR IN 60
```

The syntax of this message is terrible. It is to be read as "NEXT without FOR" error in line #60, because the computer reaches a NEXT command for which there is no accessible FOR command.

In this case we know what the problem is, but sometimes it is hard to know what is going wrong. A very valuable diagnostic tool is the TRACE command.

```
Type      TRACE
          RUN
```

It may look like a mess, but it is not. The TRACE command prints every line # as it is executed. By reading the sequence, which includes the program's print messages, you can see exactly what the computer did. In our case, it looped #10,#20,#30,#40,#50, #10,#20 and so forth, until it finished the A looping, went to #60 for the first time, and we got the error. Now correct the nesting.

```
Type      50 NEXT X
          60 NEXT A
          RUN
```

You can now study the correct looping sequence. When you have finished with TRACE, type NOTRACE and it will not trace anymore.

When writing loops, both of the following are legal options:-

```
30 FOR I=1 TO 10
...............
70 NEXT


100 FOR I=1 TO 10
110 FOR J=1 TO 5
...............
150 NEXT J,I
```

In the first case, the I is left out without penalty. (And the program runs a little faster.) In the second case, the nesting is

respected, but only one line is used.

END SESSION THREE
GOTO PROBLEM SET #3

PROBLEM SET #3

1. The kinetic energy of a moving body, K.E. = 0.5*M*V^2 , where M is the mass and V is the velocity.

Write a program to find the kinetic energy for M = 10, 20 and 30 kilograms and for velocities of 25,50,75 and 100 metres / second. Print K, M and V for each value of K. (Since the mass is in kilos and the velocity is in metres / second, the kinetic energy will be in Joules. The computer does not know this, so you must keep track of the units.)

2. Using a FOR / NEXT loop and the TAB command, write a program that prints the even numbers 2 to 38 in a diagonal across the screen.

After running this program, type SPEED =150 and run it again. The speed of execution can be set at any value from 0 to 255 (usual). It is often convenient to slow down the speed to a reading pace before listing a long program. The SPEED command does not affect the operation of programs generally. It only controls the production of screen characters. If you use SPEED to slow the screenplay, remember to reset it to 255 when you are finished.

SESSION FOUR
NUMBERS, FUNCTIONS AND VARIABLES

VARIABLES

A variable is a symbol which may take on a variety of different values. In a computer the symbol is assigned a definite location in its memory, and the current value of the variable will be stored there.

There are three types of variables in APPLESOFT. These are real, integer and string; examples of their names are, A, A% and A$, respectively. Real variables are numeric; integers, which are often used to save memory, are whole numbers and strings are alphanumeric. All the variables that we have used so far have been real variables.

The names of variables may be up to 238 characters long, but Applesoft only uses the first two characters for identification and it will treat ACCELERATION and AC63421 as the same variable. Although numbers may be used in names, the first character must always be a letter. Some names cannot be used because APPLESOFT has already taken them. A check with the Applesoft glossary at the end of this manual before assigning a name should catch most potential problems.

REAL

A real variable is numeric and fractional. In APPLESOFT, such numbers are rounded, and displayed to, nine significant figures. Values with up to 38 digits will be accepted, but they will be rounded off at the tenth digit. Scientific notation, e.g., 1.5E3=1500, is accepted by the Apple, and it becomes automatic for the Apple for numbers above 1E9 or below 1E-9.

INTEGER

Integer variables are whole numbers. In Applesoft their use is restricted to the range -32767 and +32767.

An integer variable name must have % appended, as in CD%, or FIG6%.

An integer variable cannot be used in a FOR command, or in a DEF statement (we will study DEF later this session).

STRING

A string is a sequence of characters enclosed in " ".

A string variable name must have $ appended to it, as in BAR$,Q$ and J6$.

```
Example:-  AM$="GEORGE IS 66"
           B5$="A3GH57B"

Type       NEW
           10 A$="AUGUSTA ADA BYRON"
           20 PRINT A$
           RUN
```

(This little program prints the name of the world's first programmer. Augusta Byron, daughter of the equally famous poet Lord Byron, wrote the problems for the analytical engine of Charles Babbage in the mid nineteenth century. The U.S. Dept. of Defense has named a major new language ADA in her honor.)

VARIABLE NOTES

The Apple sets all variables to zero until defined otherwise by the user. Variables are also set to zero by the commands RUN and CLEAR.

LET. This command, as in LET X=6, is never mandatory, and it will be neglected in this guide.

REM. REM statements have no effect on the program. They are remarks which explain to the reader what is happening in the program.

ARRAYS

Sometimes it is useful to store a set (matrix) of values. These could be different values of the same variable, or a group of related variables. If you intend to use an array you must assign memory space at the beginning of the program with a DIM statement. This dimension statement will specify the maximum number of elements to be stored in the array. Actually, you will get one extra element. A DIM A(20) will create the space needed for 21 elements - nos. 0 through 20. If an array is not dimensioned, space for 11 elements (equivalent to a DIM A(10) command) will be allotted by default. It is a good practice to always put in dimension statements because other languages may not provide for a default number. Also, if the array takes less than 11 elements, memory will be saved by specifying the exact number required. Do not let your programs loop back through a dimension statement since you cannot redimension an array.

The following program provides examples of loading data into real

and string arrays:-

```
Type      10 DIM A(15),B$(15)
          20 FOR X=1 to 15
          30 A(X)=X^2
          40 B$(X)=B$(X-1) + "Z"
          50 PRINT A(X),B$(X)
          60 NEXT X
```

Check, then type

```
          RUN
```

Notes.

1. B$ is loaded with Z each time, not the value of Z.

2. Values for X^2 are now stored in A() and may be recalled much more quickly than they can be recalculated.This is one of the advantages of using arrays.

3. An array may have many dimensions. A 17 by 17 matrix, for example, could be dimensioned by DIM A(17,17). This would mean that A had 17 rows and 17 columns with a total of 279 elements. (You may be more comfortable thinking of arrays as matrices.)

MATH FUNCTIONS

Many commonly used math functions are available in Applesoft. A list of them can be found in the glossary at the end of the manual. They must be used in a PRINT statement or an expression, e.g:-

```
          10 A=LOG(321.56)      (Finds the natural logarithm.)
          20 PRINT ATN(X)       (Finds the arctangent.)
          30 Y=ABS(X)           (Finds the absolute value.)
          40 PRINT SQR(B)       (Finds the square root.)
          50 S=SIN(1.5)         (Finds the sine.)
```

Trig functions always have their arguments specified in radians.

The arguments of the functions may be real numbers, integers or variables. Refer to the glossary to see the other functions that are available.

DEF

This command allows the user to define a function.

```
Type      NEW
          10 A=2.5
          20 PRINT COS(A)
          30 DEF FN Y(X)=COS(X)
```

```
        40 PRINT FN Y(A)
```
Check, then type
```
        RUN
```

Both methods will give the correct answer. The advantage of the DEF command is that it can be used for functions that are not built-in. For example:-

```
Type    30 DEF FN Y(X)=X^-3 - 3*SIN(2*X)
        RUN
```

This is a particularly useful facility when used in a loop to generate a set of function (dependent variable) values over a range of values for the independent variable (X in our example). Symbols other than Y and X may be used in the DEF command.

ROUNDING

Rounding is done using the INT function which returns the nearest whole number that is equal to or less than the real number used as the argument. For example, PRINT INT(347.8) would yield 347. The integer function, then, rounds down. Sometimes, however, we need to round up or round off. To round up, it is necessary to add 1 to the argument, e.g., PRINT INT(347.8+1) will yield 348. To round off, we need to add .5 to the argument, e.g., PRINT INT(347.8+.5) will give 348. A table of examples is provided below.

	X=5.0	X=5.49	X=5.50	X=5.99
RDG DOWN A=INT(X)	5.0	5.0	5.0	5.0
RDG UP A=INT(X+1)	6.0	6.0	6.0	6.0
RDG OFF A=INT(X+.5)	5.0	5.0	6.0	6.0

The rounding off process is shown for rounding off at the first decimal place. Rounding off at, say, the third decimal place may be done as follows:-

```
        50 PRINT INT(X*1000+.5)/1000
```

To convince yourself that this works, type,
```
        NEW
        20 X=INT(6.3125*100+0.5)/100
        30 PRINT X,INT(6.3125*1000+0.5)/1000
        RUN
```

INT, like other math functions, must be used either in a PRINT statement or in a statement assigning a value to a variable. The example shows the same number being rounded off to two and three decimal places.

If you think that you understand rounding off now, see if you can correctly predict the answer to PRINT INT(5.499*100+0.5)/100.

GENERATING RANDOM NUMBERS

Type
```
NEW
40 Y=INT(70*RND(1)+1)
50 PRINT Y
RUN
```

Line 40 generates a number randomly selected from the integers 1 to 70 inclusive. The RND function is used in many electronic games, but it is also very useful for generating samples in statistics.

The argument of RND may be any positive number without affecting the result which will always be the production of a random real number between 0 and 1.

If the argument of RND is negative, then the same random number will be generated every time that expression is executed.

The random numbers provided by the Apple are not true random numbers because they are fixed by the Apple. If you run the following program over and over, you will continuously get different random numbers. If you run it once, turn the computer off, wait a second, turn it back on and run the program again, you will get the same random numbers, viz., the first 10 numbers in the sequence.

```
10 FOR I = 1 TO 10
20 PRINT 10*RND(1)
30 NEXT I
```

Another way to get a fixed sequence is to use a negative argument. Each negative argument not only gives the same random number each time it is used, but subsequent positive arguments will follow a fixed sequence that is always the same following the same negative argument.

A RND(0) will return the last random number again.

There is a way to get a true random number from the Apple, but it is beyond the scope of this book (see Apple Orchard, Sept. '83, p76).

END SESSION FOUR:
GO TO PROBLEM SET #4

PROBLEM SET # 4

1. Use the Apple to do the following:-
   (Tip: Put 1a-1e all in one program.)

   a. Find the tangent of 1.8 radians.

   b. Find the square root of 1417.9.

   c. Find the value of e to the -3.8

   d. Generate one random whole number between 1 and 52.

   e. Find how much each person receives when $40.00 is shared among 7 people (use the rounding off process to 2 decimal places).

2. Write a program which fills an array with 12 numbers randomly chosen from the integers from 1 to 12.

3. For string variables, the command PRINT LEFT$(B$,N) will print the first N characters of the string B$. Similarly, PRINT RIGHT$(B$,N) will print the last N characters of the string B$. (PRINT MID$(B$,N) prints everything to the right of, and including, the Nth character.)

   Create a string from your full name and print your first and last names from the string separately.

SESSION FIVE: GIVE ME YOUR NUMBERS: DATA AND OTHER INPUTS

The Apple can receive data and other inputs in many ways, two of which we will consider here. One method uses internal data which is written into the program in the form of DATA statements which list the data sequentially, and which are processed by READ statements. The second method uses INPUT statements. This is an interactive technique, using external data, whereby the program requests the user to input data during the execution of the program.

READ & DATA

This program finds the average height of 10 people.

Type
```
NEW
20 FOR I=1 TO 10
30 READ D
40 S=S+D
50 DATA 5.6,5.5,6.2,6.3,5.9,6.0
60 DATA 5.3,5.4,5.7,5.7
70 NEXT I
80 PRINT "AVERAGE HEIGHT="S/10
```

Check, then type

```
RUN
```

The data here are numbers, as called for by using a real variable, D. We can write READ and DATA statements with string variables, also. For example:-

```
READ D$
DATA JOHN,MARY,MUSTAFA
```

Actually, the data values could be any combination of string, literal, real or integer symbols as long as the appropriate variables were used in the READ statement

The logic of our program is as follows. On the first loop, for I=1, the READ D command will direct the program to line #50 where the first datum, 5.6, will be read. The program then goes to line #40 and from there to line #70. In so doing, it skips right past the two DATA statements because DATA statements are always ignored, except by READ statements. From line #70, of course, the program goes to line #20 and the second loop begins. On the second loop, the second datum will be read from line # 50. The program will continue in this way until loop number six, when the last datum in line #50 will be read. On loop number seven the

first datum of line #60 will be read. The looping continues until, on the tenth loop, the last datum is read. If the number of attempts to read a datum exceeds the number of data available, an error message is generated, but the converse is not true, since the excess data will be ignored.

If you wished to add more data to this program, you would also have to change the number of loops specified. This would not be the case if you used the conditional branching method.

```
Type      NEW
          10 READ D
          20 IF D=-1 GOTO 80
          30 N=N+1
          40 S=S+D
          50 DATA 5.6, 5.5, 6.2, 6.3, 5.9, 6.0, 5.3, 5.4, 5.7,
5.7, -1
          60 GOTO 10
          80 PRINT"AVERAGE HEIGHT ="S/N
```

Check, then type

```
          RUN
```

This program works using a rogue value(-1) and a counter variable(N). The rogue value, a value no datum in a given data set will ever have, is inserted at the end of the data. When this value is read, the conditional branching statement in line #20 becomes operative and the program moves on to the next stage of execution (line #80). The counter variable keeps track of how many data are read.

If you wished to read the same set of data more than once, the RESTORE command may be used, e.g., 90 RESTORE. This will direct the next READ command to the first datum again.

The data have been listed using only one statement this time. You may list up to the line limit of 239 characters on one DATA statement, although it is a little easier to correct if the list is not too long. Another change is the inclusion of a space after each comma in the data list. This makes the data much easier to read.

INPUT

The INPUT statement requires the user to put in data, identifiers and parameters during the execution of the program.

```
Type      NEW
          10 PRINT"THIS IS A MEAN PROGRAM"
          20 INPUT"ENTER THE NAME OF THE VARIABLE TO BE AVERAGED
";A$
          30 INPUT"HOW MANY DATA VALUES DO YOU HAVE? ";N
```

```
40 PRINT "LOAD YOUR DATA: ONE PER ? AND PRESS RETURN"
50 FOR I=1 TO N
60 INPUT D
70 S=S+D
80 NEXT I
90 PRINT "MEAN "A$;" = "S/N
```

Check, then type

```
RUN
```

Remember: for the first ? you will type HEIGHT: for the second ? you will type 10. Press RETURN after each entry. Then you will load the same 10 values that you used before.

An INPUT statement allows one string in quotes at the beginning. We use it to improve the user friendliness of a program. An alphanumeric variable A$ has also been introduced to convert the program to a general averaging program that will find an average for any set of values.

Note that an INPUT statement may list more than one variable, and they need not be the same types. For example, an INPUT statement might read:-

```
95 INPUT"NAME, AGE AND WEIGHT ?";A$,Y,W
```

The single message allowed has been used for identifying all three variables. If you are very observant, you will have seen that a semi-colon follows the message, but commas separate the variables. This is very important. You can lose a lot of time on a thing like this (I lost an hour!).

END SESSION FIVE
GOTO PROBLEM SET - #5

PROBLEM SET - #5

1. Write a program that requests as input from the user the following data:-

   a. The current year (e.g., 1983).
   b. Their age.
   c. Their name.
   d. Their weight.

Then, assuming that they will gain 3% of their weight every year, design the program to tell the user, politely and by name, what their age and weight will be in 10 years (e.g., 1993).

NOTE. There is a variant of Parkinson's Law (work expands to fill the time available) which applies to computers, viz., use expands to exceed the memory available. This occurs partly because a computer can be used for so many things and by so many people. It also occurs because generating garbage is one of the things that our society does best. Sooner or later, you will view your program solutions to the problem sets as disposable. As this happens, please use the DELETE command and give your catalog a nice purge.

SESSION SIX

EDITING AND FORMATING

EDITING

Until now corrections prior to pressing RETURN have been made by backspacing and those subsequent to pressing RETURN have been made by retyping the whole line. We will now learn some other techniques.

CTRL X
To eradicate a line that you are currently typing, hold down control and hit X. This operation may be represented here in the manual (but not on the keyboard) as <CTRL-X>, or as [X].

```
Type      NEW
          10 PRINT X,3*5[X]
          LIST
```

The [X] produces a backslash and eliminates the line so that the LIST produces nothing.

DEL N,M
This command will delete line N, line M and all the lines between N and M. To delete a single line, repeat the line number.

```
Type      10 A=6
          20 B=A*COS(30)
          30 D=B^2
          LIST
          DEL 20,20
          LIST
          DEL 10,30
          LIST
```

All gone.

ESCAPE
The ESC key allows you to escape your current mode and move the cursor back "over" the program on the screen. At the desired point you may "enter" the program and make changes or corrections. This is a very useful facility, but it must be done properly. It is quite easy to end up with far more errors than you began with.

After pressing the ESC key once, the cursor may be moved around the screen without affecting either the screen image or the

memory. To do this, use the I for up, the M for down, the J for left and the K for right movements. These keys will move the cursor one position at a time, but holding down REPT as well will move the cursor very quickly.

Type
```
NEW
10 FOR X=1 TO 10
20 A=2*X + X^3
30 B$="2*X + X^3 = "
40 PRINT B$;A
50 NEXT X
```

We will now make two changes in line 10. We will set the first value of X to 2, and we will make the loop use a step of 2 rather than the default value of 1. Follow the steps carefully.

1. Press ESC once. You are now in the escape mode.
2. Press I repeatedly until the cursor is on line 10.
3. If the cursor is not over the 1 of the line number 10, then press J to move it there.
4. The cursor is now in the only position from which you can successfully enter a line.
5. Press the retype (right arrow) key once. The cursor has not moved, but you are now out of the escape mode and into line 10.
6. Press the retype key until the cursor is over the character to be changed, for us this is the 1 following the =.
7. Make the change, i.e., type 2.
8. Press the retype key until you have retyped the 10 and then type STEP 2.
9. There are two ways to get back.

    a. Press ESC once, putting yourself in the escape mode, and use the M to take the cursor back below the program. Now hit RETURN twice (once just gets you out of the escape mode) and you are back in position to continue programming.

    b. Press RETURN enough times to get you back. This works, but it is a little disconcerting because it appears to wipe out lines on the way. Actually, the screen image has its own small memory that is independent of the main memory, and the main memory is unaffected.

10. Type LIST and check the results of your editing.

NOTE. This technique is essentially a retyping of the line, making modifications as you go, which is then sent to memory to replace the old version of the line.
Remember always to enter the line to be edited at the beginning of the line and to leave it at the end. Done correctly it is efficient. If you get in difficulties, it is usually easier to resort to retyping the line.

Repeat the exercise by changing line 20 to read A=2*X^2+3*X

When editing a multi-line statement with the ESC method, a lot of undesirable spaces are generated when the retype key goes from one line to the next. The only way to avoid this is to use ESC and the K key while you are going from one line to the next.

INSERTION IN A LINE

Suppose you want to add something in the middle of a line.

Type 10 PRINT "HAPPY"

We will add TAB(10) after PRINT. First LIST 10. Then enter the line in the usual way. When the cursor reaches the space between the T and the ", stop. Press ESC again, and press I to go up one line. Type TAB(10). You will have to hit the T twice, once to get it out of ESC. (It is a good practice to use the space bar to get out of ESC, as ESC works with keys A-F, and ', as well as I,J,K and M.) Now press ESC once more, and, with the J, move the cursor back until it is over the space where you left the line - between the T and the ". Enter this space using the M. Use the retype key and take the cursor out to the end of the line as usual. Press RETURN and type LIST 10 to check your work.

NOTE. For an account of the uses of keys A to F with ESC, see the Apple II Reference Manual pp34-35.

OTHER EDITING SYSTEMS. Editing on the Apple is a little cumbersome using the built-in system described above. It is possible to purchase other editing systems for use on the Apple. These will have to be booted each time to be operational, of course.

RENUMBER. The DOS 3.3 SYSTEM MASTER disk has a renumbering program on it. If you boot this program, you will be able to load any of your Applesoft programs one at a time, renumber them, and save them back to the disk. The usual need is to get all the line numbers put into increments of ten, but RENUMBER allows any size increment to be installed over any part of the program. RENUMBER can also be used to merge two different programs.

LIST
We have already made frequent use of this command. One function of it is particularly useful here. The LIST N command will select only line N and list it. This is useful because many error messages provide the line number where the problem occurred. The statement may then be listed and edited using the ESC key method described above.

HOME
This command clears the screen (only) and puts the cursor at the top of the screen. It is often used in programs which interact a

lot with the user. It is also a nice way to get rid of a screenful of SYNTAX ERRORs. Although HOME will clear the text screen, it will not clear any of the graphics screens.

FORMATING

We have already used commas, semi-colons, TAB commands and strings to improve the readability of the output. Nothing we have so far done allows for vertical spacing arrangements. Here are two means of doing this.

Type NEW

```
10 D$="GRANNY ALWAYS ENCOURAGED ME TO STAND UP FOR
MYSELF BY SAYING, "
20 C$="REMEMBER, EVERY MIGHTY OAK WAS ONCE AN ODD NUT."
30 PRINT D$
40 PRINT C$
50 PRINT
60 PRINT D$;C$
70 PRINT:PRINT:PRINT
80 PRINT D$;
90 PRINT C$
```

Line 50 creates a line on which nothing is printed, i.e., a vertical space. Line 70 creates three vertical spaces, but the colon needs a special comment. Line 70 actually contains three commands, because the colon operates like RETURN and signals the end of a statement to the computer. Using colons, an entire FOR /NEXT loop can be placed in a single line. For example:-

```
99 FOR X=1 TO 8 : Y=X^2 : PRINT X,Y : NEXT X
```

Line 99 is perfectly acceptable in an APPLESOFT program. Other commands may be similarly condensed into a single line. Using the colon in this way results in more compact programs, as written, but the execution remains unchanged and editing is slightly more difficult. Also, the various commands under one line number cannot be separately accessed.

Line 80 ends with a dangling semi-colon. This is legal. It has the effect of putting the next PRINT output on the same line, as you will see when you run it.

OTHER SPACEY COMMANDS

VTAB,HTAB. The screen has 40 columns and 24 lines for text. HTAB 10 will place the cursor at column 10 counting from the left. VTAB 17 will place the cursor at row 17 counting from the top. These are absolute cursor moves independent of the current display and cursor position. In conjunction with a PRINT statement, these commands can place an output anywhere on the

screen. They cannot generally be used on an output to a printer. We will do only one example.

```
Type      NEW
          90 HTAB 20:VTAB 12:PRINT"HI!"
          RUN
```

SPC(N). The SPC(N) command must be used in a PRINT statement. It is very similar to the TAB command, but rather than specify the print out column absolutely, it specifies it from the column of the last printed character of the statement.

```
Type      88 PRINT "A";SPC(10)"A";SPC(15)SPC(6)"A"
```

The As will be printed at columns 1, 12 and 34 with spaces of 10 and 21 between them. The SPC command can be repeated in the manner shown. This allows spacing to be arranged beyond the usual limit of 255, and it is a feature that is not available with the TAB command.

POS(0) gives the column position of the cursor.It could be used in conjunction with a PRINT statement as follows:-

```
Type      PRINT TAB(11)"A";POS(B);SPC(17)"A";POS(C)
```

You will note that it doesn't matter what you put in the parentheses of the POS command, it still gives the current position. In this case, the first A is at column 11, the 11 takes columns 12 and 13, the SPC command adds 17 spaces, so, the next A is at column 31, as noted.

ATTENTION GETTING BEHAVIOR

INVERSE. The screen you have been staring at has green symbols on a black background. This can be reversed by using INVERSE.

```
Type      NEW
          HOME
          10 HTAB 20:VTAB 12:INVERSE:PRINT"OUR STATE!"
          RUN
```

Type NORMAL to get out of inverse. (The message used to read "Penn State", but I hope to sell a few copies at Pitt...)

FLASH. And now, what you have all been waiting for:-

```
Type      NEW
          HOME
          10 FLASH:HTAB 20:VTAB 12:PRINT"NO.1!"
          RUN
```

The audience cheers, the curtain falls, and the action moves to

PROBLEM SET - #6

To get out of the inverse and/or the flashing mode, type NORMAL.

1. Write a program which displays the word horse - flashing (horses do this naturally). Make the horse move diagonally across the screen. Use the SPEED command to make it trot at a leisurely pace.

Tips. You will need VTAB N, HTAB 2*N commands within a loop for N =1 to 17. If you include HOME in the loop, you can erase the last position of the horse. When you include FLASH and SPEED commands in a program, it is a good idea to reset the speed to 255, and the flash to normal at the end of the program so they do not become an unwanted legacy in your next task.

SESSION SEVEN
LOW RESOLUTON GRAPHICS

The Apple II+ has two graphics modes: high resolution and low resolution. The difference lies in the number of picture elements, or pixels, in the composition of the screen. In low resolution graphics the screen is composed of 1600 pixels generated by 40 columns and 40 rows. In high resolution graphics there are 280 columns and 160 rows. This results in a much larger number of pixels which, consequently, are much smaller than the low resolution graphics and therefore allow much sharper definition. Because both the number of pixels and the number of colors used place a demand on the memory, low resolution graphics can use more colors than high resolution graphics can.

In both high resolution graphics and in low resolution graphics, the bottom four lines of the screen are left in text. In high resolution, however, an option (HGR2) is available which permits full screen graphics. In neither mode is it possible to get a direct "dump" of the picture onto a plotter or a printer by a single Applesoft command as it is in the text mode. To get a printout requires a special program, or a special interface card. It is, similarly, not possible in either mode to mix text and graphics without a special program using a shape table (a stored table of shapes created graphically). Shape tables are only possible in the high resolution mode. They are not covered in this manual.

LOW RESOLUTION GRAPHICS

Low resolution can be used with 15 different colors including black and white. Black is very useful because you can erase points by replotting them in black. To get into the graphics mode all you do is type GR. However, GR automatically clears the screen (even if you are already in the graphics mode) and sets the color equal to 0, which is black. So, to be able to see anything that you draw you must set some other color. The numbers for the colors are given in BAT, p.115. The best color for both white on black and green on black screens is color 15, which is white.

COORDINATES. The screen should be thought of as a graph with the origin at the top left hand corner. the X abscissae read from left to right 0 - 39. the Y ordinate reads from top to bottom 0 - 39.

COMMANDS. There are three basic commands in low resolution

graphics:-

PLOT X,Y which plots a point at X,Y
HLIN X1,X2 AT Y which draws a horizontal line at Y from X1 to X2
VLIN Y1,Y2 AT X which draws a vertical line at X from Y1 to Y2.

```
Type      10 GR
          20 FOR X=0 TO 15
          30 COLOR=X
          40 HLIN 0,39 AT X
          50 NEXT X
          RUN
```

This programs displays the different colors, in our case, the different shades.

To return to the text mode from graphics, type TEXT. If you are in an interactive graphics program, this may not be sufficient - in which case press RESET.

```
Type      NEW
          10 GR
          20 COLOR=15
          30 HLIN 10,30 AT 10
          40 VLIN 10,35 AT 15
          50 VLIN 10,35 AT 25
          RUN
```

And there you have it, right before your very eyes - Apple pie!

```
Type      NEW
          10 GR
          20 COLOR=15
          30 FOR N=1 TO 39
          40 PLOT N,N
          50 IF N>37 GOTO 80
          60 HLIN N+2,39 AT N
          70 VLIN N+2,39 AT N
          80 NEXT N
          RUN
```

As you see, the lines in low resolution graphics are rather crude. Designs, nevertheless, are often very attractive and reminiscent of the art of the Navajo Indian nation, for example. Low resolution graphics can also be used quite successfully in games. These often have large memory requirements and this makes the lower memory demands of low resolution graphics and integer variables attractive.

BOUNCING BALL

A "bouncing ball" is an ingredient in many games. Since the

programming involved is instructive, it is reproduced below.

Type
```
NEW
10 GR
20 XN=20
30 XINC=1
40 XN=XN+XINC
50 COLOR=15
60 PLOT XN,20
70 IF XN=0 OR XN=39 THEN XINC=-XINC
80 COLOR=0
90 PLOT XN,20
100 GOTO 40
```

Check, then run this program. The ball will bounce to and fro at Y=20. It is easy to make the ball bounce off all four walls, also.

Type
```
10 GR
20 XN=20 : YN=20
30 XINC=1 : YINC=1
40 XN=XN+XINC
50 YN=YN+YINC
60 COLOR=15
70 PLOT XN,YN
80 IF XN=0 OR XN=39 THEN XINC=-XINC
90 IF YN=0 OR YN=38 THEN YINC=-YINC
100 COLOR=0
110 PLOT XN,YN
120 GOTO 40
```

In line 90, the limit of YN is set to 38 to produce an out of phase effect between YN and XN which moves the ball around the screen in a changing pattern.

Save this program. It will be used again in the next section and in the problem set.

IMMEDIATE EXECUTION PROBLEM SOLVING

If line 80 in the bouncing ball program read:-

```
80 IF XN=0 OR XN=40 THEN XINC=-XINC
```

then the program would allow XN to equal 40 before doing anything about it. A value of 40 in the PLOT command will generate an error. The program will stop and the message ? ILLEGAL QUANTITY ERROR IN 70 will be printed in the text area of the screen.

When the program stops with an error message, all the variables retain the values which they had at the moment the program stopped. Here is a useful procedure. Without leaving the graphics mode, and using the four lines of text at the bottom; type:-

```
LIST 70
```

This will produce,

```
70 PLOT XN,YN
```

Then enter,

```
PRINT XN,YN
```

This will produce something like this,

```
40                36
```

You can then see that the problem was caused by XN since 40 is an illegal quantity. Your next step is to see why the program was able to feed such a value to the PLOT command. The first places you should look is the conditional in line 80 and the increment operations in lines 30 and 40. Whereupon you will spot your mistake in a flash.

It is best to go back to the text mode for editing unless you retype the line, or it is a short line, because the LIST command will tab a double line up so that you cannot see the first line.

SOUND

The use of computers in synthesizing sound to replicate the human voice and to create music is a very active area of research, development and avocation. On a more mundane level sound is used in games and error statements. Here it will serve mainly as an introduction to the PEEK command.

The capacity of the Apple is much greater than the capacity of Applesoft. The main way of using this extra power is by adressing the memory directly, and the PEEK command is a common way of doing this.

The PEEK command will return the value stored in the memory at the specified adress, just as its companion command, POKE, will store a value at the specified adress.

Memory adresses are numbers, which for the Apple range from 0 to 49151 for the accessible memory. So, even though PEEK and POKE commands are part of the Applesoft glossary, they function like lower level assembly language because they have no meanings to the user that are drawn from English or mathematics. They simply put values in, or take values out, of numbered memory locations. The result is very hard to read, unlike higher level languages such as BASIC and FORTRAN.

If you have saved your program, type:-

```
NEW
10 B=16336
20 FOR I=1 TO 500:A=PEEK(-B): NEXT I
30 FOR I=1 TO 500:A=-PEEK(-B):NEXT I
40 FOR I=1 TO 250:FOR J=1 TO 2:NEXTJ:A=PEEK(-B):NEXT I
```

When you run this you will get several distinct tones. Only the address -16336 will activate the speaker, but the signs will vary the tone. If a second loop is nested in the first one, as in line 40, the execution is slowed and the tone lowered. Even within the loop of line 20 the tone changes, apparently due to a change in execution rate by the Apple. To get a really high pitch the program would have to be written in assembly language, which is much faster.

If the PEEK command is only executed once, the sound is so short that you probably will miss it. For the problem below a loop parameter of 5 or 10 is about right.

END SESSION SEVEN
GOTO PROBLEM SET SEVEN

PROBLEM SET SEVEN

1. Add sound to the bouncing ball program so that the ball makes a noise when it strikes the walls. Provide one noise for the side walls and a second for the top and bottom.

2. When the ball hits the side walls, make it restart from a random position on that wall.

SESSION EIGHT
HIGH RESOLUTION GRAPHICS

There are two modes for high resolution graphics on the Apple II+. The first is accessed by typing HGR and uses page 1 of memory (8K-16K). The second uses page 2 of memory(16K-24K) and is set by typing HGR2. In HGR four lines of text are left at the bottom of the screen. In HGR2 the entire screen is set in graphics. The two pages of memory mean that two pictures can be stored at the same time.

The four lines of text at the bottom of the screen in HGR can be removed and the entire screen converted to graphics with the command:-

```
POKE -16302,0
```

All the commands discussed below work equally well in both HGR and HGR2. In general, both HGR and HGR2, will be referred to as HGR.

COLOR

Eight color variations are available in high resolution graphics, including two blacks (0 and 4) and two whites (3 and 7). As in low resolution graphics, white is the best color for non-color monitors and it must be set (HCOLOR=7) immediately after setting HGR, or HGR2. Both HGR and HGR2 clear their respective screens; neither set a color.

HPLOT

Apart from the shape table commands, HPLOT is the main command in HGR. It is more powerful than PLOT, the low resolution equivalent, since it takes on the line drawing functions also.

HPLOT X,Y This will plot the point X,Y.

HPLOT X1,Y1 to X2,Y2 This will draw a line from X1,Y1 to X2,Y2.

HPLOT X1,Y1 TO X2,Y2 TO X3,Y3 TO - - - This will draw lines which join the points in the specified sequence. Remember that a program line is limited to 239 characters - and that long before you reach this limit you will have a line that will be very tough to edit. If you have a continuing shape, use a new line in which the first point is the same as the last point in the previous line.

A permissible, and very useful, variation on HPLOT is illustrated by the following partial program:-

```
50 HPLOT X(1),Y(1)
60 FOR I= 2 TO N
70 HPLOT TO X(I),Y(I)
80 NEXT I
```

The origin is at the top left hand corner of the screen.

X takes the values of 0-279 in both HGR and HGR2.

Y takes the values of 0-159 in HGR.

Y takes the values of 0-191 in HGR2.

INPUT

Data and design input for computer graphics can be effected in a number of different ways. Here we are going to use the keyboard and line by line programming. We could also input spatial information using paddles, joystick or trackball. Another method is to use a digitizing tablet. Such a tablet consists of a finely constructed electronic coordinate grid and a "pen" which, when it touches the grid, activates the circuit for that coordinate point. The point will then appear on the screen. Most tablets will also have a set of border boxes for functions that greatly imcrease the speed and accuracy of the graphics. For example, on the Apple graphics tablet, to draw a rectangle requires touching the rectangle function box and then touching two opposite corners of the desired rectangle on the grid. Even dimensioning may be done using a function box. A light pen used directly on the monitor screen is a somewhat similar method of doing computer graphics. Very sophisticated, and very expensive, computer aided design systems are now quite widespread in industry.

EXAMPLES

Type
```
HGR
HCOLOR=7
HPLOT 140,80
HPLOT 120,60 TO 140,90 TO 160,60
```

These commands plot a single point and draw an angle. The lines are irregular. This is because the screen image is composed of a finite number of discrete elements (pixels). Each pixel has a binary value, on or off (0 or 1), in the memory of the computer. Each picture means generating values for all the pixels and transmitting them to the electron beam in the cathode ray tube. As the beam sweeps the entire screen it is turned on or off according to the pixel value for the position that it is in. This is known as raster scan graphics, as opposed to vector graphics where the screen is swept by an electron beam in a continuous motion along the shape from one defined point to another, e.g.,

the oscilloscope. Vector graphics produces superb screen images, but it is not used as often as raster scan graphics, partly because the digital computer signal must be converted to an analog signal. Raster scan graphics is cheaper (even though it uses more memory) and, when plotted with a good XY plotter on paper, it will produce perfectly smooth lines (if they were defined that way) regardless of their appearance on the screen.

Type
```
NEW
TEXT
HOME

5  REM : STRAIGHT LINE DRAWING PROGRAM
10 HOME
20 HGR
30 HCOLOR=7
40 INPUT "NUMBER OF POINTS";N
50 FOR M=1 TO N
60 PRINT "POINT ";M
70 INPUT "X,Y COORDINATES? ";X(M),Y(M)
80 NEXT M
85 REM : PLOTTING
90 FOR M=2 TO N
100 HPLOT X(M-1),Y(M-1) TO X(M),Y(M)
110 NEXT M
```

Don't run this program yet. Examine the program and see if you can find any errors.

There are two mistakes. the first is using arrays without dimensioning them. The dimension statement must be put at the very beginning, in case, in a later modification, you need to loop the program back through the HOME statement. Dimension statements must not be read twice.

Type
```
8  DIM X(50),Y(50)
```

The second is less obvious. Since HGR only allows text on the bottom four lines, the interactive INPUT statements must be placed there. The best solution is simply to move the cursor there.

Type
```
37 VTAB 21
```

Now type RUN. It will be worth your while to spend a moment thinking about the coordinates that you will type in, and the sequence in which you will enter them.

Save this program. With a few modifications it can be a useful straight line drawing program. These will be added in problem set #8.

CLEARING THE SCREEN

It is often desirable to clear the screen in graphics, so this is a good place to summarize screen clearing commands.

MODE	COMMAND
Text	HOME
GR	GR
HGR	HGR
HGR2	HGR2

These commands may be placed in a program so that automatic clearing will occur in the execution of the program.

CURVES

The Apple reproduces curves quite well in HGR. Although the data for a curve could be inputted and plotted point by point, it is more usual, and much faster, to define the curve mathematically and leave the laborious part to the Apple.

Type NEW

```
5  REM : CURVE DRAWING PROGRAM
10 HOME
20 HGR
30 HCOLOR=7
40 VTAB 21
50 FOR X=0 TO 279
60 Y=0.5*X
70 IF Y>159 GOTO 90
80 HPLOT X,Y
90 NEXT X

100 FOR X=1 TO 279
110 Y=500/X
120 IF Y>159 GOTO 140
130 HPLOT X,Y
140 NEXT X

150 FOR X=0 TO 279
160 Y=60+50*SIN(X/20)
170 IF Y>159 GOTO 190
180 HPLOT X,Y
190 NEXT X
```

This program plots a straight line, an inverse and a sin curve. The relational operators in lines 70, 120 and 170 prevent illegal quantity (off the screen) errors. The inverse loop begins with 1 rather than 0 to prevent a division by 0 error. Spaces have been left to improve the readability of the program.

DISTORTION

The Apple II+ monitor provides a picture in which the vertical dimension is stretched by 10-20%. That is, if you plot a square, it will appear as an upright rectangle. It is easy to include a ratio factor in any plotting program, but this would mean that a paper (hard) copy would show a rectangle on its side. A better solution is to plot a circle at the centre of the screen and adjust the VSIZE control at the back of the monitor. This technique is not perfect either, since it reduces the screen size and some distortion may persist on other parts of the screen.

The following program plots a circle at the center of the screen. Type and run the program to examine the circularity of the image. You need not adjust the monitor.

Type

```
NEW
10 HGR2
15 HCOLOR=7
20 PI=3.14159
25 R=60
30 C=2*PI : DA=C/40
35 FOR I=0 TO C STEP DA
40 Y=R*SIN(I)
45 X=R*COS(I)
50 HPLOT X+140,Y+90
55 NEXT I
```

In this program, the center of the circle is at 140,90, which is the center of the screen in HGR2. The radius is 60. There are 40 plotted points. Each angular increment, DA, is equal to 2*pi divided by forty. The angles are in radians. (A circle may also be drawn with lines using a polygon with a large number of sides. See the program on Page 67.)

DATA

Data is often easier to understand when it is organised graphically. If we have experimental data involving two variables X and Y, we can plot the data with a program such as the following. Study, but do NOT type:-

```
NEW
10 REM : X,Y DATA PLOT
20 DIM X(50),Y(50)
30 HOME
40 HGR
50 HCOLOR=7
60 VTAB 21

70 REM : DATA INPUT
80 INPUT "NUMBER OF DATA PAIRS? ";N
90 FOR K=1 TO N
100 PRINT "FOR DATA PAIR NUMBER ";K
```

```
110 INPUT "X,Y VALUES ARE? ";X(K),Y(K)
120 NEXT K

130 REM : HIGHEST AND LOWEST VALUES
140 FOR K=1 TO N
150 IF X(K)>XH THEN XH=X(K)
160 IF Y(K)>YH THEN YH=Y(K)
170 IF X(K)<XL THEN XL=X(K)
180 IF Y(K)<YL THEN YL=Y(K)
190 NEXT K

200 REM : SCALING
210 RX=XH-XL
220 RY=YH-YL

270 REM : PLOTTING
280 FOR K=1 TO N
290 XP=(X(K) - XL)*279/RX
300 YP=(Y(K) - YL)*159/RY
305 YP=159-YP : REM INVERTING Y AXIS
310 HPLOT XP,YP
320 NEXT K

330 REM : AXES
335 AY=159 : AX=0
340 IF XL<0 THEN AX=ABS(XL*279)/RX
350 IF YL<0 THEN AY=159-ABS(YL*159)/RY
360 HPLOT AX,0 TO AX,159
370 HPLOT 0,AY TO 279,AY
380 END
```

The scaling routine used here makes sure the range of experimental data will fit on the HGR screen. You will be asked to explain how it does this in Problem Set #8.

The data plotting program is constructed from a logical sequence of little programs. Computer programs are typically written in this manner. It is usually a straightforward task to add to, or to modify, them. The data plotting program, for example, would benefit from additional routines which 1, print out the main parameters (e.g., XH), 2, place user specified reference marks on the axes and 3, use a shape table to put units on the axes and a title to the drawing. None of these routines would be hard to do.

END SESSION EIGHT
GOTO PROBLEM SET - #8

PROBLEM SET - #8

1. Write down on paper an explanation of the plotting program at the end of Session Eight.

2. Modify the straight line drawing program in two of the following ways:-

a. Provide an option for the user to decide whether or not to join the first and last points.

Tip.
```
120 INPUT"DO YOU - - -? 1 FOR YES, 2 FOR NO? ";Q
130 IF Q>1 GOTO ?
```

b. Provide an option for the user to add to the drawing.

c. Provide an option for the user to do a new drawing.

d. When all options are declined, tell the user that the party's over.

SESSION NINE
INTRODUCTION TO INTERMEDIATE PROGRAMMING

At the end of the last session you saw how a program is built of a series of small routines. These routines are the building blocks of computer programs and writing programs this way, as opposed to a continuous stream of statements , is known as structured programming. Each routine is a little program with a self contained logical sequence performing a specific function. Any given program can usually be extended by adding further routines. Most programs can in turn be made part of a yet larger program. For example, the data plotting program of Session Eight could easily be part of a program written to monitor an experiment.

Computer programming, then, means identifying the operations to be performed, and the sequence in which they must be performed. It is a mistake to try to write the whole program at once unless it is very short. The following steps may be used as a guide.

1. Decide on the overall purpose(s) of the program.

2. Outline the sequence of operations necessary to achieve it. It is usually better to have many simple routines rather than a few complex ones.

3. Temporarily set aside writing routines which you know from experience will be straightforward to write and which are not at the analytical core of the program.

4. Write one routine at a time and test it before writing the next one. Write your routine to handle the simplest case first. Then, adapt it to accept the most common cases. Consider the special problems last.

5. When writing the second, and subsequent, routines, make sure that they are compatible with each other. For example, a variable may have a negative value in one routine without penalty, but, when transferred to another routine, it may be an illegal quantity.

6. When the analytical core of the program has been written, write the set of introductory statements. Examples of these would be to clear the memory and the screen, set the mode and color, set base values and dimension arrays. The program can then be tested as a whole.

7. Add any additional INPUT, PRINT and REM statements that will make the program more user friendly.

8. When your program is working perfectly for you, give it to

somebody else to test. It will not work for them.

9. When your program works for you and for ten other people, it may be considered foolproof. It will not, of course, work for everyone. Human diversity is a wonderful thing, and it is a constant source of creativity in computer programming, as elsewhere.

FLOW COMMANDS

We have already used the flow commands FOR / NEXT, GOTO and IF / THEN, and seen how powerful they are. Happily, there are more of them, and we will now study GOSUB, ON and GET. Both of the first two are used to branch from the main program, go to a specified routine, and, in the case of GOSUB, return back to the same place in the program.

GOSUB

The name GOSUB is short for "go to subroutine", and subroutine means a routine, which, while necessary to the program, serves the analytical core without being a part of it. Subroutines are typically placed at the end of a program from where they are called upon as required. It is often possible to save processing time, and certainly program writing time, by writing a frequently used routine just once and using it as a subroutine on demand. Be sure to put an END statement at the end of the main program so that the subroutines will not be inadvertently executed.

The address of the statement following GOSUB in the main program is stored by the execution of GOSUB, and the RETURN command at the end of the subroutine used will send the program to that address in the main program.

EXAMPLE. Consider the problem of finding how many different groups of 4 people could be obtained from a class of 32. This is a combination problem, the general formula for which is:-

$$C=N!/(R!(N-R)!)$$

The problem is specific, but we will write a general solution program and input our specific data.

Type

```
NEW
10 REM : COMBINATIONS
20 HOME
30 INPUT"WHAT IS THE TOTAL NUMBER? ";N
40 PRINT:INPUT"WHAT IS THE SUBGROUP SIZE? ";R

45 REM : FIND FACTORIALS
50 X=N
```

```
60 GOSUB 200
70 NF=F
80 X=R
90 GOSUB 200
100 RF=F
110 X=N-R
120 GOSUB 200
130 NR=F

140 REM : SUBSTITUTE IN EQUATION
150 C=NF/(RF*NR)
160 PRINT TAB(123)"COMBINATIONS OF ";R;" FROM ";N;" =
";C
170 END

200 REM : FACTORIAL SUBROUTINE
210 F=1
220 FOR J=1 TO X
230 F=J*F
240 NEXT J
250 RETURN
```

In this program it is necesary to find the factorial of a number three times. The routine for doing this is written just once, but used three times by GOSUB commands. Since it is written only once, program variables must be translated into the subroutine variables before GOSUB, and subroutine variables must be translated into program variables upon return to the main program. This procedure becomes slightly more complex where arrays are involved. Before continuing, check this program over and run it. Be aware that factorials generate very large numbers. The limit of the Apple is reached with 34!

ON

The ON statement is a traffic control device. the controls may be program generated or, as in the following example, be user initiated through input statements.

Study, but do not type:-

```
10 PRINT "MAKE YOUR SELECTION FROM 1-4":PRINT
20 PRINT "INPUT DATA - 1"
30 PRINT "GET A MEAN - 2"
40 PRINT "GET A DATA PRINT OUT - 3"
50 PRINT "GET A DATA PLOT - 4"
60 INPUT "YOUR CHOICE IS?";N
70 ON N GOTO 350,150,200,300

150 REM : MEAN ROUTINE                    (Not shown)
190 GOTO 10
200 REM : PRINT OUT ROUTINE               (Not shown)
240 GOTO 10
300 REM : DATA PLOT ROUTINE               (Not shown)
```

```
340 GOTO 10
350 REM : DATA INPUT ROUTINE                    (Not shown)
390 GOTO 10
```

This partial program allows the user to choose between several different available routines. The value input for N will decide which routine is chosen. If 3 is chosen, the ON command will direct the program to the third line number in its list. In an ON command GOSUB may be used instead of GOTO. If GOSUB is used, a RETURN command is needed at the end of the subroutine. The program will then return to the line following the ON command.

The ON command has many uses. In graphics, for example, it could be used to move the cursor with key board entry (1 for up, etc.) for plotting a point or placing a shape.

GET

Although similar to INPUT in that it requires a value to be input, the value of the GET command is to control the program flow externally. It takes an input of only a single character. It does not require the RETURN key to be pressed after the entry is made. If an alphanumeric variable is specified, a single letter must be entered. If a real or an integer variable is designated, then a single number must be input. The following is an example which could be placed in the combinations program.

```
84 PRINT "PRESS ANY LETTER TO CONTINUE"
85 GET A$
```

This puts execution of the program into the control of the user at line 85. It is, of course, quite feasible to make use of the values so obtained in the program itself. Once the value is input, but not before, the program will continue.

FINALE

For the finale you will be taken for a ride. This program has been written to incorporate many of the techniques and commands that you have learned in one program. Your final task will be to get this program running, and to turn in a written explanation of the program.

Type

```
NEW
5 DIM X(20),Y(20)
10 HGR2
20 GOSUB 1000

30 REM : TAKING OFF
40 FOR J=20 TO 260 STEP 20
50 V=J^2/500
60 V=185-V
70 SLOPE=J/250
```

```
80 HCOLOR=7
90 GOSUB 500
100 FOR P=1 TO 200:NEXT P
110 HCOLOR=0
120 GOSUB 500
130 NEXT J

140 REM : LANDING
150 TEXT:HOME
160 PRINT"LANDING VELOCITY: PRESS A NUMBER FROM 1 (SLOW)
TO 4 (FAST)"
170 GET S
180 HGR2
190 LV=10*S

200 FOR I=260 TO 20 STEP -LV
210 V=185-I^2/500
220 J=279-I
230 SLOPE=-I/250
240 HCOLOR=7
250 GOSUB 500
260 FOR P=1 TO 200:NEXT P
270 IF I=20 GOTO 400
280 HCOLOR=0
290 GOSUB 500
300 NEXT I
400 END

500 REM : PLOTTING SHAPE
510 HPLOT J,V
520 FOR K=2 TO 13
530 XP=J+X(K)
540 YP=V+Y(K)-SLOPE*X(K)
550 HPLOT TO XP,YP
560 NEXT K
570 RETURN

1000 REM : LOADING DATA FOR SHAPE
1100 FOR I=1 TO 13:READ X(I):NEXT I
1200 FOR I=1 TO 13:READ Y(I):Y(I)=-Y(I):NEXT I
1300 RETURN
2000 DATA 0,-4,-2,-11,-15,-17,-15,-16,-12,3,19,6,0
3000 DATA 0,0,8,8,5,5,8,13,10,13,10,8,0
```

This program takes coordinate points for a sketch of a plane and repeatedly plots and erases the shape to give the impression of motion. There are several features of the program worth noting.

1. In line 1200 the Y ordinates are inverted because the Apple measures Y downwards. This only adjusts for the shape, not for the position on the screen. The position is adjusted for in lines 60 and 210.

2. The flight path is along a parabola, Y=X^2/500. The slope at

any point is dy/dx, or X/250.

3. The adustment in line 540 for plotting the shape on the necessary slope is only a crude approximation. This is why the shape distorts as the slope increases. If the shape were plotted correctly, the necessary calculations would slow the plotting down to the point where the motion effect would be lost.

4. Lines 100 and 260 provide a pause for viewing the shape.

SESSION TEN : AN INTRODUCTION TO COMPUTER GRAPHICS

In this session several basic concepts in computer graphics will be introduced. These are digital shape definition, placing, translation, scaling, rotation, clipping and viewports.

The style of this session will differ from that of the previous sessions. It is about twice as long and it will focus on the programming routines that accomplish the transformation algorithm being considered rather than on providing complete programs. The task for the reader is to put together complete programs which incorporate one or more of the algorithms. The ability to assemble programs is the last step in the process of acquiring basic literacy in programming.

SHAPE DEFINITION

In its final representation a shape is expressed the same way in computer graphics that it is in traditional graphics, that is, spatially. However, until it is expressed in its final form, and whenever it is changed in any way, a shape in computer graphics is expressed numerically or digitally. It may also be expressed as a math function such as Y=SIN(X). When expressed numerically a shape is usually expressed in coordinates. Three dimensional shapes are readily handled in computer graphics but the representation will always be a two dimensional projection. In this session we will only work with two dimensional shapes.

Coordinate representation of a shape can be written as X1,Y1; X2,Y2 and so on, but it is usually more convenient to store the data in arrays. Three ways of doing so are given below. The shape provided is an arrowhead. Using the coordinates given in the DATA statements, sketch the shape on paper before you continue.

Study these three methods, but do not type them in unless you are a compulsive typist.

METHOD I

```
50 READ N
60 FOR I=1 TO N: READ X(I): NEXT I
70 FOR I=1 TO N: READ Y(I): NEXT I
1000 DATA 5
2000 DATA 0,4,2,4,0
3000 DATA 0,4,0,-4,0
```

METHOD II

```
50 READ N
```

```
60 FOR I=1 TO N: READ X(I),Y(I): NEXT I
1000 DATA 5
2000 DATA 0,0,4,4,2,0,4,-4,0,0
```

METHOD III

```
50 READ N
60 FOR I=1 TO N
70 FOR J=1 TO 2
80 READ XY(I,J)
90 NEXT J,I
1000 DATA 5
2000 DATA 0,0,4,4,2,0,4,-4,0,0
```

Since working with a two dimensional array is probably new for you, here is a PRINT routine which will allow you to check it.

```
100 FOR I=1 TO N
110 FOR J=1 TO 2
120 PRINT XY(I,J),
130 NEXT J
140 PRINT
150 NEXT I
```

The dangling comma in 120 and the PRINT in 140 combine to produce the data in very readable coordinate pairs. A plotting routine could be:-

```
200 HPLOT XY(1,1),XY(1,2)
210 FOR I=2 TO N
220 HPLOT TO XY(I,1),XY(I,2)
230 NEXT I
```

THE SHAPE ORIGIN

For the arrowhead shape given above the origin of the shape is at its tip. It is very important in computer graphics to be aware of where the shape origin is. For example, the arrowhead cannot be plotted as described without getting an illegal quantity error because of the -4 value. This does not mean that the shape origin of the arrowhead should be changed by, say, adding +4 to all the Y values. By the end of this chapter you should be able to appreciate that choosing a shape origin and placing the shape on the screen are quite separate processes.

PLACEMENT

Placing a shape on the screen involves adding a value PX, say, to all the X values of the shape and a value PY, say, to all the Y values. The shape origin is a very useful guide in choosing PX and PY because since the shape origin is 0,0 you will be, in effect, placing the shape origin at PX,PY on the screen. The other thing to bear in mind is the size of the shape. To avoid

illegal quantity errors the following two criteria must be met. PX and PY must be at least as large as any respective negative X and Y values. And PX and PY when added to the largest respective positive X and Y values must be no greater than the screen limits (e.g., in HGR these would be X=279 and Y=159).

The placing coordinates may be used to create new arrays which can be subsequently plotted:-

```
300 FOR I=1 TO N
310 PX(I)=X(I)+PX:PY(I)=Y(I)+PY
320 NEXT I
```

Or they may be used in a plotting routine:-

```
300 X=X(1)+PX:Y=Y(1)+PY
310 HPLOT X,Y
320 FOR I=2 TO N
330 X=X(I)+PX:Y=Y(I)+PY
340 HPLOT TO X,Y
350 NEXT I
```

Or they may be placed directly in the plotting statement:-

```
300 HPLOT X(1)+PX,Y(1)+PY
310 FOR I=2 TO N
320 HPLOT TO X(I)+PX,Y(I)+PY
330 NEXT I
```

TRANSLATION

Translation is the process whereby a shape is moved from one position to another. Although this effect could be achieved by repeated placing routines, it is more convenient to keep the two procedures separate. In general, a shape may be moved by an amount TX in the X direction and TY in the Y direction. TX and TY may both be either negative or zero. As with placing, however, their values must be chosen such that illegal quantities are not generated. There are several ways in which a translation may be programmed.

The placing arrays which were generated in the first option for placement above could be continuously modified for each new translation (and subsequently plotted):-

```
400 FOR I=1 TO N
410 PX(I)=PX(I)+TX:PY(I)=PY(I)+TY
420 NEXT I
```

Or the plotting routine could be directly modified:-

```
400 FL=0
410 FOR I=1 TO N
420 X=X(I)+PX+TX:Y=Y(I)+PY+TY
```

```
430 IF FL=1 THEN HPLOT TO X,Y
440 IF FL=0 THEN HPLOT X,Y:FL=1
450 NEXT I
```

Other arrangements are not hard to think of. A direct modification of the HPLOT statement being the most obvious.

Whichever option you choose keep in mind whether you want to translate continuously by modifying the last position, or translate relative to an original position. The first option above is for a continuous translation. For translation from a fixed position we need to protect the integrity of the original placement arrays by creating new ones for the current plot:-

```
400 FOR I=1 TO N
410 CX(I)=PX(I)+TX
420 CY(I)=PY(I)+TY
430 NEXT I
```

PROBLEM 10.1

Take a shape of your own choosing and establish the coordinates that define it. You may use the squared paper at the end of the graphics workbook to do this. Write a program that reproduces the shape in six different places on the screen.

SCALING

Scaling is the transformation used to make a shape larger or smaller. It is done through the use of scale factors. Since a different scale factor can be used for the X and Y values, the same procedure can be used to distort the shape with either an X or a Y bias.

The basic algorithm for scaling is very simple:-

```
500 FOR I=1 TO N
510 NX(I)=FX*X(I)
520 NY(I)=FY*Y(I)
530 NEXT I
```

FX and FY are the scale factors for X and Y respectively. When FX and FY are not equal distortion will result. Note that the algorithm shown here protects the original shape matrix by creating new ones. This means that the scaling is not continuous from modification to modification but always applied to the original shape. You may choose to use a continuous algorithm by using the same arrays on both sides of 510 and 520.

The shape origin is very important in scaling. Consider the following alternative DATA statements for representing the same shape.

```
1000 DATA 5
```

```
2000 DATA 0,4,2,4,0
3000 DATA 0,4,0,-4,0


1000 DATA 5
2000 DATA 100,104,102,104,100
3000 DATA 100,104,100,96,100
```

Both sets of coordinates describe the same arrowhead shape. In the first set the tip is at 0,0, in the second the tip is at 100,100. If we increase the size of the arrowhead by a factor of 5 using our scaling algorithm, we will get very different plotting prospects from the two sets of data.

The first set may be plotted by adding PX and PY such that when added to the original data times five will not generate illegal quantities. If PX and PY are chosen once to place the shape well clear of the edges of the screen, repeated variations of scale may be carried out.

The second set of data have been transformed into unplottable quantities. To plot them would require subtracting approximately 500 from both the X and the Y arrays. Any technique for generalizing the subtraction process effectively converts the second data set back to the first. It is better therefore to use the first data set.

What has happened to the second data set under the scaling transformation is both a translation and a scaling. If the shape origin is outside the shape, then the distance it is outside also becomes scaled - hence the translation. Generally, only the shape origin itself remains unmoved during the scaling transformation. As a rule of thumb it is best to keep the shape origin in or at least on the shape itself.

PROBLEM 10.2

Write a program that will plot the arrowhead shape mid-screen. Make your program interactive so that you can independently control the scale factors for X and Y.

ROTATION

As with scaling the location of the shape origin is very important because the shape origin is the point about which the shape will be rotated. The effect of a rotation on a shape depends only on the point of rotation relative to the shape and the angle through which it is rotated. The algorithm we use for rotation is developed below and it uses the shape origin as the point of rotation.

From FIG. 1 we can establish the following relationships by

trigonometry.

$$D = X1/COS(A) = Y1/SIN(A)$$

$$X2 = D*COS(A+B) = D*COS(A)*COS(B) - D*SIN(A)*SIN(B)$$
$$Y2 = D*SIN(A+B) = D*SIN(A)*COS(B) + D*COS(A)*SIN(B)$$

Substituting for D yields,

$$X2 = X1*COS(B) - Y1*SIN(B)$$
$$Y2 = X1*SIN(B) + Y1*COS(B)$$

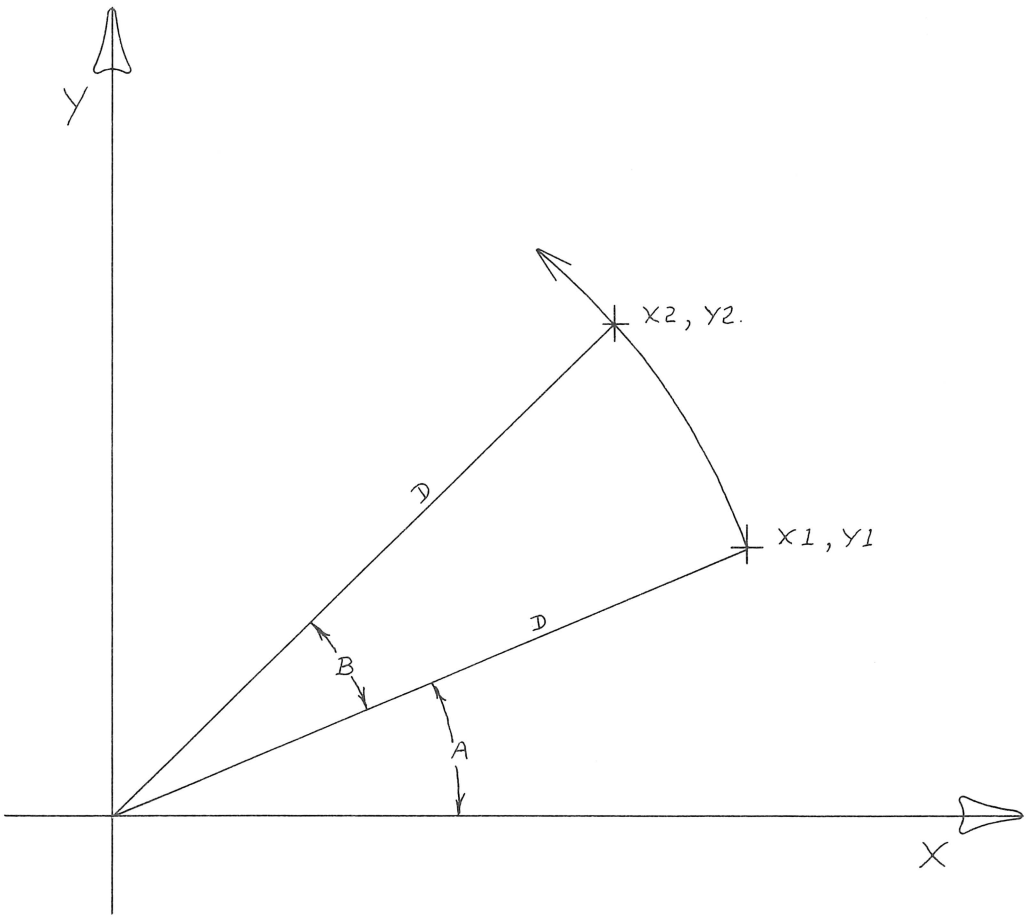


FIG. 1

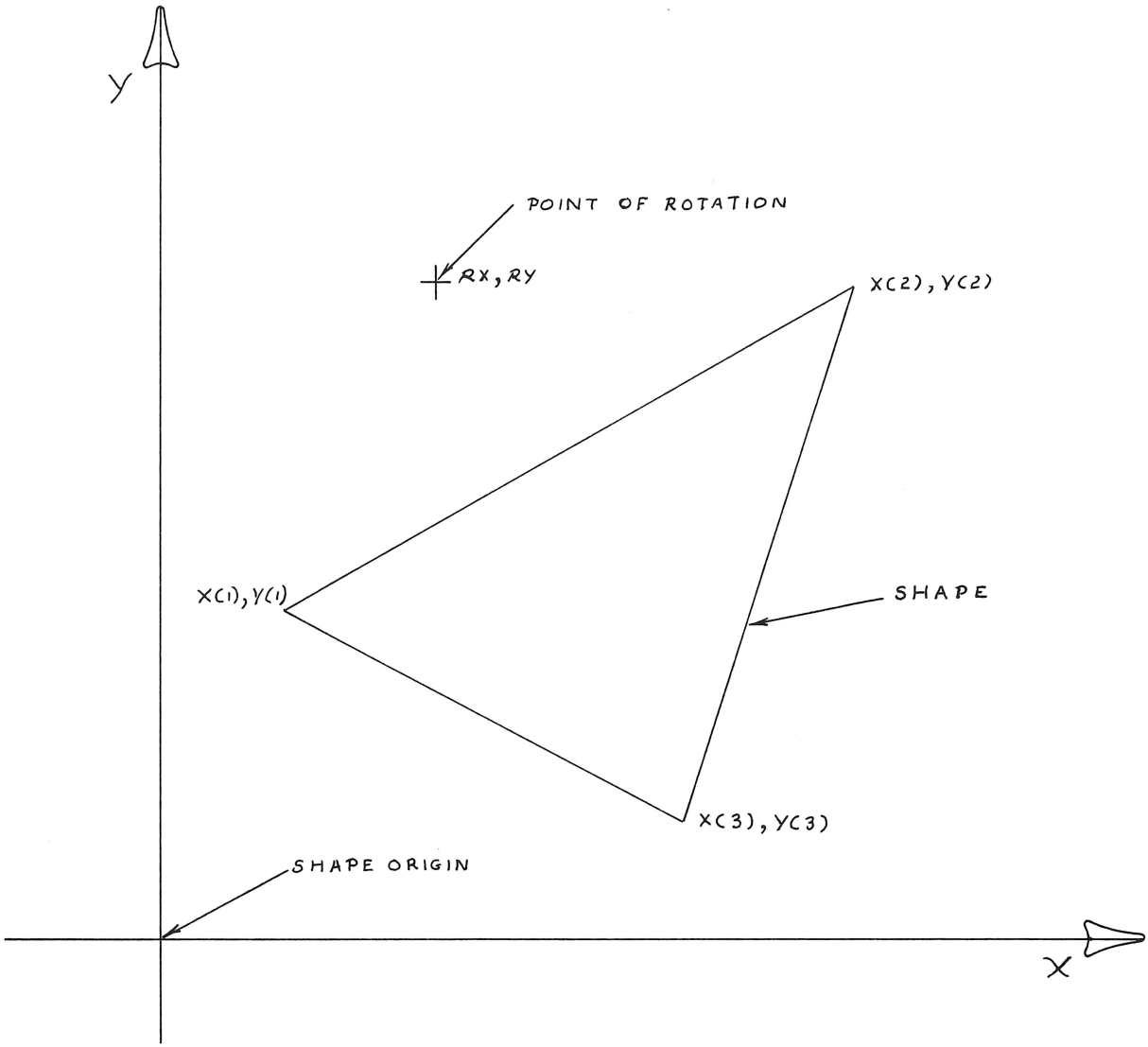


FIG. 2

We now have the coordinates of point 2 described in terms of the coordinates of point 1 and the angle of rotation between them. Assuming rigid body rotation, all points on a shape that is rotated must be rotated through the same angle. Therefore, if all points on a shape are defined relative to the same origin / point of rotation, the algorithm that we have derived is applicable to all the points on a shape during a rotation.

If we desire to rotate the shape about any point other than the shape origin, the shape coordinates must be redefined with a new shape origin that corresponds to the desired point of rotation. The redefinition is like the placement process and simply involves applying a translation of RX and RY, say, to the shape coordinates. RX and RY may take any values, but keeping the shape on the screen becomes increasingly difficult as RX and RY become larger.

It is important to remember that RX and RY represent a temporary redefinition of the shape for the purposes of rotation only and the translation must be reversed at the end of the transformation.

If you refer to Fig. 2, you will see that the RX and RY values are defined relative to the original shape origin and subtracted from the original shape coordinates in order to shift the origin to the point of rotation. After the rotation RX and RY are added back to reverse the translation. (Obviously, the actual values for RX and RY may be positive or negative.)

The algorithm may be programmed as follows:-

```
600 INPUT " ANGLE OF ROTATION? ";AR
610 AR=AR*3.14159/180
620 INPUT " POINT OF ROTATION RX,RY? ";RX,RY
625 FOR I=1 TO N
630 X(I)=X(I)-RX
640 Y(I)=Y(I)-RY
650 TEMP=X(I)*COS(AR)-Y(I)*SIN(AR)
660 Y(I)=X(I)*SIN(AR)+Y(I)*COS(AR)
670 X(I)=TEMP+RX
680 Y(I)=Y(I)+RY
685 NEXT I
```

Line 610 converts the angle from degrees to radians. The use of the temporary variable is because X and Y are interactive and it prevents Y being calculated from the new X value rather than the old value. This routine will continuously modify working from the last position of the shape. It may be rewritten to always modify from a fixed position of the shape by using new arrays and thereby protecting the original arrays.

Remember that when you plot the shape after a rotation transformation you will still need to add placing coordinates PX and PY. And, of course, don't add them in before the rotation!

PROBLEM 10.3

Write a program with a shape of your choice that will place the shape anywhere on the screen and rotate it about any point. (Tip: keep your shape small.)

CLIPPING AND VIEWPORTS

Clipping is a process whereby the drawing produced is contained in a legal domain. In HGR, for example, the values of X must be >=0 and <=279, and the values of Y must be >=0 and <=159. Attempts to use values outside of these limits will result in the error message:-

```
? ILLEGAL QUANTITY ERROR
```

There are two ways to avoid this problem. One is by making sure, through scaling and placing, for example, that the program will never produce an illegal quantity. The second method is clipping off the illegal quantities. Clipping is a general procedure which for given boundaries will check values produced by the program for plotting to ensure that only legal values are fed to the HPLOT command. Clipping, then, does not necessarily prevent the generation of values that are illegal for plotting, but it does necessarily prevent the attempt to plot them.

Consider the following circle drawing program:-

```
10 HGR2:HCOLOR=7
20 R=120:DA=2*3.14159/100
30 FOR I=1 TO 100
40 Y=R*SIN(A)
50 X=R*COS(A)
60 IF X<0 OR X>279 THEN GOTO 90
70 IF Y<0 OR Y>191 THEN GOTO 90
80 HPLOT X,Y
90 A=A+DA
100 NEXT I
110 END
```

Lines 60 and 70 clip the X and Y values that exceed the legal boundaries of HPLOT by skipping the execution of HPLOT for those values.

LINE CLIPPING. It is relatively easy to clip point by point plotting. If we need to clip part of a line, the problem becomes more complex (see Myers, Roy E. Microcomputer Graphics. Addison-Wesley, 1982). However, the point clipping method can be used on whole lines - if a little crudely. The technique used is to check the end point to which the line is to be drawn. If that

point lies out of the legal domain, the line is not drawn. The next line will not be drawn either. A line will not be drawn again until both its end points lie within the legal domain. This approximate process may often produce rather crude results. It can also be very satisfactory where the lines are short, as in the polygon / circle drawing program below when the number of sides is large.

```
10 T=2*3.14159
20 INPUT "HOW MANY SIDES DOES YOUR POLYGON HAVE? ";N
30 C=COS(T/N) : S=SIN(T/N)
40 INPUT " RADIUS? ";R
50 X=R : Y=0 : CX=140 : CY=96
60 HGR2 : HCOLOR=7
70 FL=0
80 FOR I=0 TO N
90 SX=X+CX : SY=Y+CY
100 IF X<0 OR X>279 THEN FL=0 : GOTO 140
110 IF Y<0 OR Y>191 THEN FL=0 : GOTO 140
120 IF FL=0 THEN HPLOT SX,SY : FL=1
130 HPLOT TO SX,SY
140 XN=X*C-Y*S : Y=X*S+Y*C : X=XN
150 NEXT I
```

You should be able to recognise line 140 as a rotation transformation complete with a temporary variable, XN.

VIEWPORT. A viewport is the name given to the legal drawing domain. This domain may be imposed, as it is by the screen size in the example above, or it may be designated by the programmer to be some limited area of the screen. If the viewport is less than the full screen, it may be plotted as a rectangle using HPLOT in the usual way.

PROBLEM 10.4

Write a high resolution graphics program that establishes a rectangular viewport 80 high and 120 wide. Draw lines to delineate the boundaries of the viewport. Next, draw a circle centrally placed in the viewport with a radius of 50. Use a clipping procedure to ensure that no values are plotted outside the viewport.

*** END OF SESSION TEN ***

END OF THE LITERACY SEQUENCE

*****

DOS: THE FILE MANAGEMENT SYSTEM

The management of files is important in almost all uses of microcomputers because files represent the permanent memory for programs and text files. (Text files are files in which information such as numerical data or prose is stored in strings.) The major file management system for the Apple II is DOS, and there is much more to DOS than the RUN, LOAD and SAVE commands that were covered in the Introduction. In this chapter we will cover some further aspects of DOS, viz., file protection, the control D option, exec files, sequential files and random access files.

PROTECTION

It is mentioned several times in earlier chapters that it is very easy to lose a program. This section is devoted to the ways in which you can prevent this from happening.

LOCK/UNLOCK. If a file is locked, it may not be written on - although you may run the file as often as you like. This means that if you tried to save a program to a locked file name you would be prevented, and the message FILE LOCKED would appear on the screen.

To lock a file you simply type LOCK VECTOR in the immediate execution mode and press return. To unlock the file, type UNLOCK VECTOR. (VECTOR is an example of a program name). If the file is locked, it will appear in the catalog with an asterisk beside it, e.g.,

```
*A 003 VECTOR
```

As with other DOS commands the locators for slot and drive must be added where necessary, e.g., LOCK VECTOR,S6,D2.

SECRET NAMES. Control characters, generated by holding down control and pressing a letter, may be included in a program name. This will make it almost impossible for anyone to run or change the program unless they know what the control characters are. Although attractive as a security feature, if you forget what you did, you will join the ranks of the excluded. (Some of the more sophisticated copying programs can identify control characters.)

DISK PROTECTION. We can also write protect a disk. If you hold a floppy disk in your hand so that you can read the label the right way up, you will notice a notch cut in the right hand side of the disk about an inch from the top. If this notch is covered, it is impossible to write ,e.g., save programs to, the disk. Many boxes of diskettes come with little adhesive paper tabs intended for write protecting disks.

There is another way of inhibiting writing on disks. The locators

discussed so far have been for slot and drive numbers, Sn,Dm. A volume number may also be added, and the order does not matter. The following is acceptable:-

```
SAVE VECTOR D2,V46,S6
```

The volume number works as a disk identifier. It may be any integer from 1 to 254. The number is assigned with the INIT command when the disk is initialized, e.g., INIT HELLO,S6,D1,V101. If no volume number is given in the INIT command, a default number of 254 is given. Since most people do not give a volume number to their disks, and because a lot of HELLO programs bring up the volume number, you will often see this volume number on the screen. The value of using the volume number is that you must specify it in order to use anything on the disk, so you have a check for saving the right program to the right disk.

VERIFY. This is a method of checking that your program has been properly saved. It is easy to do. Type VERIFY VECTOR and add locators if necessary (and press return). If there is a problem, an I/O ERROR message is produced. Otherwise no message is given. There is no evaluation of the program itself, the only check is on the process by which the program was saved and, subsequently, on any damage to the program such as a scratch on the disk. The method uses a numerical sum that is always calculated at the time of the save and recorded with the program.

COPYING. A very important means of protection is to have back-up copies of programs and disks. Having two back-up copies of your more valuable disks is not excessive. Always make the back-up at the first opportunity.

The DOS 3.3 SYSTEM MASTER disk for the Apple II has two copy programs. COPYA will copy an entire disk including the DOS which saves you the trouble of initializing the copy-to disk. FID will copy one program at a time. (FID is in binary so you must BRUN FID.) It also provides several other features, the most valuable of which is one that will tell you how much space you have left on a disk.

There are many copying programs on the market, some of which are quite sophisticated. A major aspect of this sophistication is the ability to copy copyrighted disks that have various anti-copy features. Copying copyrighted disks without the permission of the owner of the copyright is illegal. This is true even if the disk is outrageously expensive and very easy to copy.

THE CONTROL D OPTION

A CONTROL D is the signal sent to the computer when the CONTROL key is held down and then the D key is pressed. This signal allows you to put DOS commands in a program. For example, the HELLO program may be written to automatically provide a catalog

listing.

```
10 D$=CHR$(4) : REM CHR$(4) IS CONTROL D
20 HOME:VTAB 1:HTAB15
30 PRINT "TRON'S DISK"
40 VTAB 3
50 PRINT D$;"CATALOG"
60 END
```

This program is an extension of the HELLO program given in the introduction. The extension provides an automatic catalog when the disk is booted. This is done in line 50. In this line the D$ is a signal that the next characters are a DOS command. D$ is CONTROL D which may be entered in two ways. The above method uses the ASCII Character Code number for CONTROL D. This code is an industry standard which provides numeric values for the characters entered at the keyboard. A list of these values is provided in The Applesoft Programming Reference Manual, pp138-9. The other method is to set D$ = "" and put a CONTROL D between the quotes. The problem with this method is that you cannot be sure that it is there until the program works because control characters do not print out on the screen.

The CONTROL D option is very important in making EXEC and text files. Before commenting on these, a command that is very useful for monitoring the operations performed on text files will be introduced.

MON C,I,O . The MON command may be used with any combination of the C, I and O parameters, in any order, and as long as at least one is specified. Implementation of the C parameter will provide a print on the monitor of the commands being issued to the disk. The I and O parameters will display on the monitor the input and the output from the disk, respectively. For example, MON C would display all the RUN commands in the EXEC program listed below. The MON C,I,O command is terminated by NOMON C,I,O.

EXEC FILES

An EXEC file is a program that incorporates a series of DOS commands which may, for example, RUN several different programs and / or READ and WRITE to several different text files. Since we will cover text files in the next section, the EXEC example here will use RUN only, but even to do this we will have to make a text file.

```
10 D$=CHR$(4)
20 PRINT D$;"OPEN DEMO"
30 PRINT D$;"WRITE DEMO"
40 PRINT "RUN BALL4"
50 PRINT "RUN CURVE2"
60 PRINT "RUN LOGO"
70 PRINT D$;"CLOSE DEMO"
SAVE DISPLAY
```

```
RUN DISPLAY
EXEC DEMO
```

If you type the above you will create an Applesoft program called DISPLAY which, when run, will create a text file called DEMO. This text file cannot be implemented by a RUN command, but it can by an EXEC command. When you type EXEC DEMO, the file will be executed as a series of DOS commands and the three programs BALL4, CURVE2 and LOGO will be run sequentially (if they are accessible on the disk). The contents of the text file DEMO are the three RUN commands. Other DOS commands could be listed also - preferably loaded one per print statement, because that guarantees that each DOS command will be followed by the necessary RETURN. The RUN statements are loaded into the file by PRINT commands because WRITE redirects PRINT output from the monitor to the disk.

This creation of an EXEC program serves as an introduction to text files, particularly the clumsy quality of the programming involved. The awkward nature of it results from DOS and Applesoft being quite separate language systems. The key linkage between the two is the CONTROL D signal. Text files are important enough to make the effort required to work with them well worth while.

TEXT FILES

There are two types of text files, SEQUENTIAL and RANDOM ACCESS. The latter uses a system whereby all data is recorded within records of a fixed length. The records are numbered sequentially which makes the retrieval and insertion of individual data easy, e.g., addresses in a mailing list. Sequential text files record information in fields of lengths which vary according to the data - each RETURN signal marking the end of a datum.

SEQUENTIAL TEXT FILES. The data that you enter into a text file will be in the form of strings. These may be words, numbers, or a mixture. Each string may have up to 239 characters.

```
  5 DIM L$(15)
 10 D$=CHR$(4)
 20 PRINT D$;"OPEN LEXICON"
 30 PRINT D$;"WRITE LEXICON"
 40 PRINT "AARDVARK"
 50 PRINT "AARDWOLF"
 60 PRINT "ABACA"
 65 PRINT D$;"OPEN LEXICON"
 70 PRINT D$;"READ LEXICON"
 80 FOR J=1 TO 3
 90 INPUT L$(J)
100 NEXT J
110 PRINT D$;"CLOSE LEXICON"
120 FOR J=1 TO 3
130 PRINT L$(J)
140 NEXT J
```

This program creates a text file called LEXICON, places three names in it, loads those three names from it into an array, L$, and prints them out as a check that the file was correctly created. The file is then closed. The file is written to by the PRINT statements following the WRITE command. The file is read by an INPUT statement which, following a READ command, takes input from the disk instead of from the keyboard. Note that in line 65 an OPEN command is used. This places the READ which follows at the beginning of the file, otherwise the access would remain at the end of the third entry, "ABACA".

In programs using text files the disk replaces the role of the user and PRINT output goes to the disk rather than to the monitor for the user to see. Similarly, input is taken from the disk rather than from the user via the keyboard.

To add to this file do not use OPEN or you will put the new strings at the beginning of the file, replacing what is already there. Use the APPEND command.

```
  5 DIM L$(15)
 10 D$=CHR$(4)
 20 PRINT D$;"APPEND LEXICON"
 30 PRINT D$;"WRITE LEXICON"
 40 PRINT "ABACUS"
 45 PRINT D$;"OPEN LEXICON"
 50 PRINT D$;"READ LEXICON"
 60 FOR J=1 TO 4
 70 INPUT L$(J)
 80 NEXT J
 90 PRINT D$;"CLOSE LEXICON"
100 FOR I=1 TO 4
110 PRINT L$(J)
120 NEXT I
```

If you want to substantially change a file, it may be best to delete the old file before saving the new one if the same name will be used. Otherwise, if the old file is longer than the new file, the surplus will not be written over and it will remain in the file. Since a file must be OPEN to delete it and since each DOS command will cancel the last, a delete sequence will look like this:-

```
30 PRINT D$;"OPEN LEXICON"
40 PRINT D$;"DELETE LEXICON"
50 PRINT D$;"OPEN LEXICON"
```

Finally, it should be noted that all the usual locators may be used with the DOS commands in these text file programs.

RANDOM ACCESS FILES. These differ from sequential files by having data stored in records of a fixed length, L, and an entry record number, Rn. The parameters are attached to the the text file

commands. Otherwise there is little difference in the syntax used.

```
  5 DIM R$(50)
 10 D$=CHR$(4)
 20 FOR I=1 TO 50
 30 INPUT "ENTER RESISTOR VALUE";R$(I)
 40 NEXT I
 50 PRINT D$;"OPEN RESISTORS,L10"
 60 FOR J=1 TO 50
 70 PRINT D$;"WRITE RESISTORS,R";J
 80 PRINT R$(J)
 90 NEXT J
 95 PRINT D$;"OPEN RESISTORS"
100 FOR K=1 TO 50
110 PRINT D$;"READ RESISTORS,R";K
120 INPUT R$(K)
125 PRINT R$(K)
130 NEXT K
140 PRINT D$;"CLOSE RESISTORS"
```

This program first solicits input for 50 values from the user in the usual way and stores it in the array R$. It then opens a text file named RESISTORS with a fixed record length of 10 bytes. Next, it writes the contents of R$ into RESISTORS numbering each datum sequentially as it enters it. The data are then retrieved from the file to show how this is done. In the loops which load and retrieve data from the text files the delimiters are 1 and 50. Other delimiters may be used. This means that data can be loaded and retrieved to and from anywhere in the files quite easily. This is why random access files are useful. Note that after a READ command the PRINT output will go to the monitor, thus saving a loop in the verification sequence at the end of the program.

A SELECT BIBLIOGRAPHY FOR FURTHER READING

PROGRAMMING

Coan, James S. BASIC APPLE BASIC. Rochelle, New Jersey: Hayden Book Co., 1982.

Higgins, David A. PROGRAM DESIGN AND CONSTRUCTION. Englewood-Cliffs, New Jersey: Prentice-Hall, 1979.

Lewis, William E. PROBLEM SOLVING PRINCIPLES FOR BASIC PROGRAMMERS. Rochelle, New Jersey: Hayden, 1981.

Mateosian, Richard. INSIDE BASIC GAMES. Berkeley, California: Sybex, 1981.

GRAPHICS

Hearn Donald and M. Pauline Baker. MICROCOMPUTER GRAPHICS. Englewood-Cliffs, New Jersey: Prentice-Hall, 1983.

Myers, Roy E. MICROCOMPUTER GRAPHICS. Reading, Massachusetts: Addison-Wesley,1982.

Rogers, David F. and J. Alan Rogers. MATHEMATICAL ELEMENTS FOR COMPUTER GRAPHICS. New York: Mcgraw-Hill, 1976.

PASCAL

Koffman, Elliot B. PASCAL A PROBLEM SOLVING APPROACH. Reading, Massachusetts: Addison-Wesley, 1982.

Lewis, William E. PROBLEM SOLVING PRINCIPLES FOR PASCAL PROGRAMMERS. Rochelle, New Jersey: Hayden, 1981.

FILES

Miller, David. APPLE FILES. Reston, Virginia: Reston Publishing, 1982.

Swanson, Paul. MICROCOMPUTER DISK TECHNIQUES. Peterborough, New Hampshire: Byte/Mcgraw-Hill, 1982.

ASSEMBLY

Inman, Don and Kurt Inman. APPLE MACHINE LANGUAGE. Reston, Virginia: Reston Publishing, 1981.

Wagner, Roger. ASSEMBLY LINES: THE BOOK. North Hollywood, California: Softalk Publishing, 1982.

MICROCOMPUTER PERIODICALS
(With an Apple emphasis)
1983

The following information was collected during the summer of 1983 and is intended to be accurate for that time. However, no guarantee that the prices, or other data, are correct is given.

APPLE, the personal computer magazine and catalog
Regis McKenna Publications
1800 Embarcadero Rd., Palo Alto, CA 94303
-quarterly since 1979
price: unknown

Apple Orchard
International Apple Core, Inc.
908 George St., Santa Clara, CA 95050
-monthly since 1979
price: $24.00/year

BYTE, the small systems journal
McGraw-Hill Publications Co.
70 Main St., Peterborough, NH 03458
-monthly since 1975
price: $18.00/year

Call-A.P.P.L.E.
Apple Puget Sound Program Library Exchange
21246 - 68th Ave. S., Kent, WA 98032
-monthly since 1977
subscription by A.P.P.L.E. membership only

Compute!
Compute! Publications, Inc.
P.O. Box 5406, Greensboro, NC 27403
-monthly since 1978
price: $25.00/year

Creative Computing, magazine of personal computer applications and software
Ziff-Davis Publishing Co., Consumer Division
One Park Ave., New York Morrisontown, NY 10016
-monthly since 1974
price: $24.97/year

Data Cast, the user's reference publication for major micro software systems
Wireless Digital, Inc.
345 Swett Rd., Woodside, CA. 94062
-bimonthly since 1981
price: $18.00/year

Info World
Popular Computing, Inc.
530 Lytton, Palo Alto, Ca 94301

-weekly since 1979
price: $31.00/year

Interface Age, business oriented microcomputer magazine
16704 Marguard, Box 1234, Cerritos, CA 90701
-monthly since 1975
price: $18.00/year

International Journal of Mini and Microcomputers
Acta Press
Box 2481, Anaheim, CA 92804
-3 issues/year since 1976
price: $58/year

Journal for Microcomputer Applications
Academic Press, Inc. Ltd.
111 Fifth Avenue, New York, NY 10003
-quarterly since 1982
price: unknown

MICRO, the 6502 journal
Micro Ink Inc.
34 Chelmsford Ave, Box 6502 Chelmsford, MA 01824
-monthly since 1977
price: $15.00/year

Microcomputer Applications
Acta Press
Box 2481, Anaheim, CA 92804
-3 issues/year since 1980
price: $56/year

Micrographics Today
National Micrographics Assocation
8728 Colesville Rd., Silver Spring, MD 20910
-monthly since 1967
price:free to members of assoc.

Mini-Micro Computer Report
EDP News Services, Inc.
7620 Little River Turnpike, Annandale, VA 22003

Nibble, the reference for Apple computing
Nibble-Micro SPARK
Box 325, Lincoln, MA 01773
-8 issues/year since 1980
price: $19.95/year

Personal Computing
Hayden Publishing Co. Inc.
50 Essex St., Rochelle Park, NJ 07662
-monthly since 1977
price: $14.00/year

Popular Computing

McGraw-Hill Publications Co.
Box 590, Martinsville, NJ 08836
-monthly since 1979
price: $8.50/year

Portable Computer
Miller Freeman Publications
500 Howard St., San Francisco, CA 94105
-bimonthly since 1983
price : $15.97/year

Softside, your basic software magazine
Softside Publications
515 Abbot Drive, Broomail, PA 19008
-monthly since 1978
price: $24.00/year

UTILITY AND SYSTEM

As discussed in the introduction and in the chapter on DOS, locators must be appended as necessary for slot, drive and volume for CATALOG, SAVE, LOAD, RUN, RENAME, LOCK, UNLOCK and DELETE. You may need all, some or none of the locators. They may be appended in any order. An example using all three is RUN VECTOR,S6,D1,V35.

CATALOG provides a directory of all the files stored on the disk that is currently booted, or the disk in the location to which the command is sent.

CONT continues a program which has been interrupted by STOP, END, or ctrl-C.

ctrl-C used in immediate mode to halt program execution (except interactive) or listing.

ctrl-S can be used as a toggle switch for stopping and starting listing and printed output.

DELETE X erases file X from the disk.

END halts program execution.

LOAD X loads the BASIC file X into memory after clearing memory.

LOCK X protects program X against accidental change or deletion. LOCKed files are shown in the CATALOG with an *.

NEW erases current program from memory.

NOTRACE turns off TRACE.

PEEK(X) Looks at and retrieves the contents of memory location X.

POKE X,18 puts the value 18 in memory location X. The new value must be between 0 and 255.

RENAME X,Y changes a disk file's name from X to Y.

ctrl-RESET/RESET stops any APPLESOFT command unconditionally and immediately.The program is left intact although variable values are maintained as of the break-off (they will be cleared by a RUN command).

RUN runs the program currently in memory.

RUN 315 executes program starting at line 315.

RUN X runs program X from the disk.

SAVE ABC saves the program in current memory to the disk and gives it the name ABC.

STOP halts execution and reports in which line (its own).

TRACE lists each line number as it is executed. (Used in debugging.)

UNLOCK X unlocks previously LOCKed file X to allow change or deletion.

FORMAT AND EDITING

CLEAR sets all variables to zero.

DEL X,Y deletes form line X to line Y, inclusive.

FLASH makes the monitor output flash.

HOME clears the screen and puts the cursor at the top.

HTAB X moves the cursor to horizontal position X in text mode (1 to 40 across the screen left to right). Only works in deferred execution mode.

INVERSE sets the monitor output to black on white or black on green.

LIST lists the program in memory.

LIST 60,350 lists from line 60 to line 350, inclusive.

LIST 100- lists from line 100 to the end of the program.

NORMAL turns off FLASH and INVERSE.

POS(0) returns the horizontal position of the cursor (1 to 40). Used in PRINT statements.

REM XYZ for writing program comments; ignored by the program.

SPC(X) used in PRINT statements only; puts X spaces between the last item printed and the next.

SPEED=X sets character output rate from 0 (slowest) to 255 (fastest and normal).

TAB (X) used in PRINT statements only; moves the cursor to position X (1 to 255) measured from the beginning of the line that the cursor is currently on (40 positions per line).

VTAB Y moves the cursor to vertical position Y in the text mode (1 to 24 top to bottom).

STRINGS AND ARRAYS

ASC(A$) returns the ASCII code for the first character of A$.

CHR$(X) returns the ASCII character whose code is X.

DIM A(I,J,K) sets maximum dimensions for A; reserves memory space for I+1,J+1,K+1 real elements, starting with A(0,0,0).

DIM A$(I,J) sets maximum subscripts for A$, which may contain I+1,J+1 string elements, each up to 255 characters.

LEFT$(A$,N) returns the first N characters of A$.

LEN(A$) returns the number of characters in A$.

MID$(A$,N,M) returns M characters of A$, beginning at Nth character.

RIGHT$(A$,N) returns the last N characters of A$.

STR$(X) creates a string out of the numeric value of X.

VAL(A$) gives the numeric value of A$ up to the first non-numeric character.

INPUT/OUTPUT

DATA X,"Y",Z establishes list of data elements, real, integer or string, that can be used by READ statements.

GET A$ calls for a keyboard input of a one-character value for A$; does not need RETURN key. Takes real, integer or string values.

INPUT "X=";X calls for keyboard input for a multi-character value for X followed by RETURN. Prompts with a ?. Takes real, string or integer variables. Allows one string after INPUT which will precede the prompt.

IN#6 takes future input from peripheral device in slot #6, instead of from keyboard (IN#0).

PRINT "Y="Y prints string Y= and value of variable Y on the screen. Semicolons concatenate printed items, commas separate items into three tab fields. The symbol ? also means PRINT.

PR#6 sends output to the peripheral device in slot#6, instead of to the monitor screen (PR#0)

READ X assigns the next DATA value to X. Takes real, string and integer variables (single and array).

RESTORE reestablishes DATA statements so that READ will start at the beginning again.

FLOW CONTROL

FOR I=M TO N STEP P beginning statement in a FOR / NEXT loop. Positive, negative and decimal numerals may be used for M, N and P.

GOSUB 55 directs the program to the subroutine at line 55.

GOTO 190 directs the program to line 190.

IF X=5 THEN A=B. If the assertion X=5 is true, then the execution continues and A is given the value of B. If the assertion is false, then execution jumps to the next numbered line. A statement linked after and to a conditional by a : will only be executed if the condition is true.

NEXT X marks the end of a FOR...NEXT loop. The x is optional.

ON X GOSUB 190,55,75 directs the program to the subroutine at the line number in the Xth position in the list.

ON X GOTO 190,55,75 directs the program to the line number in the Xth position in the list (if X=2, goes to line 55).

RETURN is the last command of a subroutine; it directs the program to the statement following the GOSUB that was last executed.

GRAPHICS

COLOR=X sets color (0 to 15) for next plotting in low-resolution graphics.

GR sets low-resolution graphics (40 x 40); bottom 4 lines are text. Also clears the low resolution graphics screen and sets the color to 0 (black).

HCOLOR=X sets color (0 to 7) for plotting in high-resolution graphics.

HGR sets page 1 high-resolution graphics (280 x 160), bottom 4 lines are text. It also clears the HGR screen.

HGR2 sets page 2 high-resolution graphics (280 x 192). It also clears the HGR2 screen.

HLIN X1,X2 at Y draws a horizontal line in low-resolution

graphics from X1,Y to X2,Y.

HPLOT X,Y places a dot at horizontal coordinate X and vertical coordinate Y in high-resolution graphics. X is from 0 to 279; Y is from 0 to 159 (HGR) or 191 (HGR2). The origin is at the top left corner, so Y is measured downwards.

HPLOT X1,Y1 TO X2,Y2 draws a line from X1,Y1 to X2,Y2 in high-resolution graphics. The command may be continued to other points TO XN,YN and so on.

PLOT X,Y place a dot in low-resolution graphics at horizontal coordinate X and vertical coordinate Y. X and Y are from 0 to 39. The origin is at the top left corner, so Y is measured downwards.

VLIN Y1,Y2 AT X draws a vertical line in low-resolution graphics from X,Y1 to X,Y2.

MATH FUNCTIONS

Note: all the trig functions take their arguments in radians.

ABS(X) the absolute value of X.

ATN(X) the arctangent, in radians, of X.

COS(X) the cosine of X.

DEF FN A(X)=X*24/(X^2-6) defines a function FNA. Subsequently FNA() may be used in place of the defined expression. In this example, FNA(3)=24.

EXP(X) gives e (2.718289) to the power X.

INT(X) the largest integer less than or equal to X.

LOG(X) the natural logarithm of X.

RND(1) produces a random real number from 0 to 0.99999999.

RND(0) repeats the last random number.

RND(-3) returns 4.48217179e-08. Each different negative argument will produce a different random number. After a RND with negative argument, RND with a positive argument will follow a fixed sequence unique to that negative argument.

SGN(X) gives the sign of X, or 0 if X=0 (signum function).

SIN(X) the sine of X.

SQR(X) the positive square root of X.

TAN(X) the tangent of X.

THE FIRST FEW BYTES

Richard F. Devon

KENDALL/HUNT
PUBLISHING COMPANY
Dubuque, Iowa

B 403250 01
ISBN 0-8403-3250-5